Dissertation

Gale Duality, Decoupling, Parameter Homotopies, and Monodromy

Submitted by

Matthew E. Niemerg

Department of Mathematics

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2014

Doctoral Committee:

Advisor: Daniel J. Bates

Patrick Shipman
Chris Peterson
Chihoon Lee

Abstract

Gale Duality, Decoupling, Parameter Homotopies, and Monodromy

Numerical Algebraic Geometry (NAG) has recently seen significantly increased application among scientists and mathematicians as a tool that can be used to solve nonlinear systems of equations, particularly polynomial systems. With the many recent advances in the field, we can now routinely solve problems that could not have been solved even 10 years ago. We will give an introduction and overview of numerical algebraic geometry and homotopy continuation methods; discuss heuristics for preconditioning fewnomial systems, as well as provide a hybrid symbolic-numerical algorithm for computing the solutions of these types of polynomials and associated software called `galeDuality`; describe a software module of `bertini` named `paramotopy` that is scientific software specifically designed for large-scale parameter homotopy runs; give two examples that are parametric polynomial systems on which the aforementioned software is used; and finally describe two novel algorithms, *decoupling* and a heuristic that makes use of monodromy.

Acknowledgements

The list of people whom I would like to thank cannot possibly be enumerated. To begin with, I would like to thank my loving family: parents, siblings, and large extended family. Next, I would like to thank all of my teachers whom I have admired and who have inspired me throughout the years. My formal mathematical and computer science training did not begin until I started at Eastern Illinois University after high school but were it not for those who pushed me early on, I would not be where I am today. I would like to thank Duane Broline, Keith Wolcott, Andrew Mertz, and Patrick Coulton, amongst the many faculty members at EIU, who taught me to enjoy mathematics for its own sake, as both an intellectual challenge and the beauty behind it all.

I have nothing but gratitude for my advisor, Dan Bates, especially for his pick-your-own-adventure mentality. I do not believe I could possibly have had a better mentor or a friend. Thanks to Dan Brake for our wonderful discussions on Numerical Algebraic Geometry and thoughts on proper programming protocol. Also thanks to Brent Davis for countless talks on our discipline, particularly about the latest developments in the field. Thanks to all of my collaborators for helping me to understand and for all the great work you do.

Finally, I would like to thank the Divine for wisdom, inspiration, and the strength to complete this dissertation.

Table of Contents

List of Tables

List of Figures

CHAPTER 1
Introduction

1.1. Why solve systems of equations?

Solving polynomial systems of equations is, in general, a difficult task. Applications arise in many areas of engineering and the sciences, and the solutions of a system can be used to describe observable phenomena that scientists find useful. For example, solutions to some systems may describe optimal values given certain constraints and conditions. In addition, the solutions can also provide valuable insight to the mathematician. Equations can be divided into two classes: linear and nonlinear. Nowadays, linear problems are well-understood with many well-studied solution techniques. The solution of linear problems is only hindered by hardware limitations. On the other hand, nonlinear problems pose a challenge, especially if the system is multivariate. We will (almost) exclusively discuss in the duration of this dissertation multivariate systems. We will briefly explore problems that arise from antiquity as well as ones that scientists and mathematicians encounter in today's world.

1.2. Nonlinear Polynomial Problems from Antiquity to Today

Thousands of years ago, mathematicians began considering nonlinear univariate polynomials and attempted to find analytic solutions. Iterative methods were developed for approximating the positive roots of a quadratic equation, such as $x^2 = 5$ (i.e. $x^2 - 5 = 0$) as early as the 1st century A.D. by Heron of Alexandria [45]. Unfortunately, the mathematics at that time was not sophisticated enough to recognize the difference between positive and negative roots, not to even mention complex roots! It was not until 1545 when Gerolamo

Cardano published solutions for univariate cubics and quartics that the first reference to imaginary numbers appears (although, the wording is not the same as today's terminology). It is amusing to note that Cardano had asked Niccolò Tartaglia to show him his method for solving cubics of the form $ax^3 + bx + c = 0$. Tartaglia did so with reluctance, on the promise that Cardano would not publish the results. Later, Cardano's student, Ludovico Ferrari, discovered the algebraic solution of the quartics, which required Tartaglia's method for solving cubics of the prior form. However, once Cardano learned that Scipiano del Ferro had also discovered the formula for the cubic, he did not hesitate to publish the results of del Ferro and Ferrari. Cardano freely acknowledges these events in his book [22].

The main multiplicative rules of complex numbers were first given by the Italian mathematician Rafael Bombelli in 1572 [19]. In 1629, the French mathematician Albert Girard acknowledged the existence of both negative and complex roots [36]. Eight years later, Renè Descartes published his "rule of signs" [26] to determine the number of real solutions to a univariate polynomial. For more on the history of solving polynomials which occurred during the Renaissance, the reader is referred to the archival resources maintained by O'Connor and Robertson on the University of Saint Andrew's website [65].

Progress, in the sense of necessary mathematical theory, was made for the multivariate setting over the next several centuries; a rekindled interest did not occur until the 1970's when Askold Georgevich Khovanskii began developing the theory of fewnomials [49]. This theory generalizes Descartes' rule of signs to nonlinear multivariate polynomials more formally and is the necessary background upon which §4 relies.

1.2.1. Differential Equations. Many nonlinear problems today arise from first-order ordinary differential equations. These solutions are the critical points for a given

system and can help determine minima and maxima. When dealing with a large number of variables, finding analytic solutions to the system becomes prohibitive, and one normally turns to numerical techniques. One then wants to know how to find all the solutions of a system of ODEs. In addition, classifying the stability of the equilibria points is another common task. If the ODE is polynomial, one can use the tools of Numerical Algebraic Geometry to find numerical approximations to the equilibria of the ODE in question as well as determine stability type. Some examples are the 2-point boundary problems described in [1].

Often, differential equations are of a form in which they have parametrized coefficients. The software package `paramotopy` [4] was originally developed to attack these types of problems. In the applied world, parameter values are largely unknown, and as such, exploring the parameter space can yield information about the relationship between variables. Sometimes the particular system needs to be solved at multiple parameter values of interest. Many applications, such as in biochemical reaction networks [24] and finding the equilibria in certain string theoretic models [44], also have systems with parametrized coefficients.

In certain settings of phenomenological string theory, finding all the vacua of a potential equation is a much-studied problem. The vacua correspond to local minima of the potential equation. In many of these string theoretic models, the coefficients of the potential can be considered as parameters. One can take the first order derivatives of the potential and compute all of the equilibria points. The question is then to determine the number of solutions that are also completely real and the type of stability associated with them, amongst other statistical information like variance and mean.

In section 6.2, we will explore an example in biochemical reactions, where the reaction rates can be considered as parameters, namely because the reaction rates are largely unknown. We will see how through the use of the `paramotopy` software, we are capable of locating regions of parameter space where we can find parameter values where the number of real solutions increases.

In addition, we will also consider in §6.3 an example related to rock magnetism where the goal is to find the stable states of magnetic fields. We employ a novel algorithm called *decoupling*. We will also give some insight as to when *decoupling* can or cannot be used. For more information on *decoupling*, see §3.1. For additional examples, see §6.3.2 and §3.1.2.

Partial Differential Equations provide another source of polynomial systems. Many physical systems from electrodynamics, fluid flow, and heat are described using PDEs. Finding a closed form analytic solution to a general PDE is difficult; however, finding some or all numerical solutions for PDEs which are completely polynomial in nature is a task for numerical algebraic geometry. Using well-known finite difference methods, one can treat the terms in the finite difference as variables of a polynomial system. Research has been done with incorporating NAG, specifically homotopy continuation, methods to finding numerical solutions of PDEs [38, 39, 40]. In addition, a one-to-one correspondence exists between the monomials and the differential operators. By considering the Jet Variety (more on varieties can be found in §2.3) and the techniques used for attaining more constraints of a PDE [67, 68, 85], one could also use various methods to find only the real points or components of these varieties [15, 54].

CHAPTER 2
Numerical Algebraic Geometry

2.1. Introduction

Numerical algebraic geometry is a field which, in essence, takes the rich theoretical structure of algebraic geometry and introduces numerical and other computational techniques to find all solutions of polynomial systems in a relatively new way. In this section, I will walk you through the history of Numerical Algebraic Geometry, describing its earliest thoughts, ideas, and some of the obstacles that the field encountered during its development.

In the late 1970's, Zangwill and Garcia [35] and Drexler [28] independently proved theorems that homotopy continuation methods could possibly be used to find all the isolated solutions of polynomial systems. The concept was that one could construct a *homotopy path* or *path* from all of the solutions of a polynomial system one knows how to solve, known as the *start system*, to all of the solutions of a polynomial system that one wants to solve, known as the *target system*.

One could, in principle, use predictor methods (Euler, Runge-Kutta-5, etc.) coupled with a multivariate Newton correction, transform the solutions of the start system to the solutions of the target system. This transformation of solutions is known as a *homotopy*. For more homotopies and the the general algorithm of *homotopy continuation* see §2.2.

During the next few years, much theory was developed regarding what choices of homotopies yield all the complex solutions [23, 34, 59]. See §2.2 for "good" choices of a homotopy.

At this time, the advancements were made mainly in choosing an appropriate *start system* (see 2.2) in such a way that reduced the number of starting paths [53, 58, 61] as most systems are deficient [52].

Around this same time, a novel technique proposed by Wright [84] was to track the homotopy paths in projective space. By tracking the paths in projective space, we are guaranteed the arclengths of all the paths are of a finite length. Morgan proposed a somewhat different transformation [60].

Beyond the development of how to construct homotopies that work generically and the crucial insight of being able to track in projective space, numerical algebraic geometry and homotopy continuation techniques began to crop up in a wide variety of applications and disciplines ([18, 25, 55, 62, 64]).

One of the main hurdles of numerical algebraic geometry was ensuring numerical stability. More recent work has succeeded in showing that for general systems numerical stability can be achieved. Methods were developed to overcome these issues, namely by using higher precision. Big number libraries (GMP, MPFR, etc.) are in active development that allow for arithmetic to be done at higher precision levels. The downside to this is that the computations in higher precision are performed at the software level and not at the hardware level. This fact has the drawback of slowing down the computations (i.e. software arithmetic and multiplication is slower than arithmetic and multiplication built into the arithmetic logic units on CPUs). In order to overcome an ill-conditioned systems, one could, conceivably, solve a polynomial system at a fixed higher precision. However, the level of precision necessary is not known *a priori*, and one risks the chance of not setting the precision to a high enough value. At the same time, for many systems, only some paths are ill-conditioned and, for the most part, not along the entire path. (If this is the case, one could perform a scaling techniques, SCLGEN or SCLCEN, described in both [56] and [27]). Because higher precision at the software level is not necessary for every path and every portion of a path, it

becomes beneficial to move back to the hardware level after moving beyond ill-conditioned zones. This is known as Adaptive Multiprecision (AMP) and has been the subject of many articles (see [9]). Because it is faster to perform calculations at the hardware level, one may try to avoid increasing precision at the software level as much as possible. A new algorithm proposes to do precisely this and is briefly discussed in §3.5 and more thoroughly presented in §7.

Now, over a set of measure zero in the parameter space of the coefficients, multiple paths will not have the same solution value, i.e., multiplicities will not exist for any given time t along the the path. Essentially, the so-called γ trick is used in such a way that guarantees that path-crossings do not occur. For more on the γ trick, see §2.2.2. However, for many applications, the target system has real coefficients which may have solutions with multiplicity bigger than 1. In order to deal with such situations when multiple paths come together for a target system, end-game methods were developed in order to handle these numerical instabilities [8, 9, 47]. Once the end-game calculations are complete, all the zero-dimensional solutions are known, or at least the ones that are not considered to be tracking to the projective point at infinity that is determined by a user-defined value.

Prior to the turn of the century, numerical algebraic geometers became not only concerned with zero-dimensional solutions, but also the representation of positive-dimensional components of polynomial systems. Numerical algebraic geometry provides ways to use numerical techniques to extract algebraic geometric data about the solution set of a polynomial system [7]. This exact geometric data gives information regarding the number, degree, and dimension of positive dimensional components. Computing this information is known

as finding the numerical irreducible decomposition and a slew of algorithms have been developed in order to do so efficiently, most notably [71], [73], [70], [72], and [51]. More on positive-dimensional varieties and systems that have positive-dimensional components can be found in §2.3.

2.2. Homotopy Continuation

Let $H(z;t) : \mathbb{C}^n \times (0,1] \mapsto \mathbb{C}^n$ such that:

(1) for any choice of $t \in (0,1]$, $H(z;t)$ is a system of polynomials;

(2) given a start system, $\mathcal{S} = \{s \mid H(s;1) = 0\}$, with D solutions, there is a smooth map $p_i(t) : (0,1] \mapsto \mathbb{C}^n$ satisfying $p_i(1) = s_i \in \mathcal{S}$; and

(3) p_i is smooth on $(0,1]$, and for each $t^* \in (0,1]$:

   (a) $\nexists j,k \in \mathbb{Z}$ such that $1 \leq j < k \leq D$ and $t^* \in (0,1]$ such that $p_j(t^*) = p_k(t^*)$; and

   (b) the points $p_j(t^*)$ are smooth isolated solutions of $H(z;t^*)$.

A homotopy that satisfy these regularity conditions is a "good" homotopy.

Given a square polynomial system $g : \mathbb{C}^n \mapsto \mathbb{C}^n$ and a similar system $f : \mathbb{C}^n \mapsto \mathbb{C}^n$ with known and easily computable solutions, one deforms g (the *start system*) into f (the *target system*) by constructing a homotopy

$$H(z;t) = \gamma t g(z) + (1-t) f(z)$$

and using basic predictor / corrector methods as t varies from 1 to t_0, where t_0 is some predetermined value at which an endgame is performed (see [8, 9, 47]). We let $\gamma \in \mathbb{C}$ be

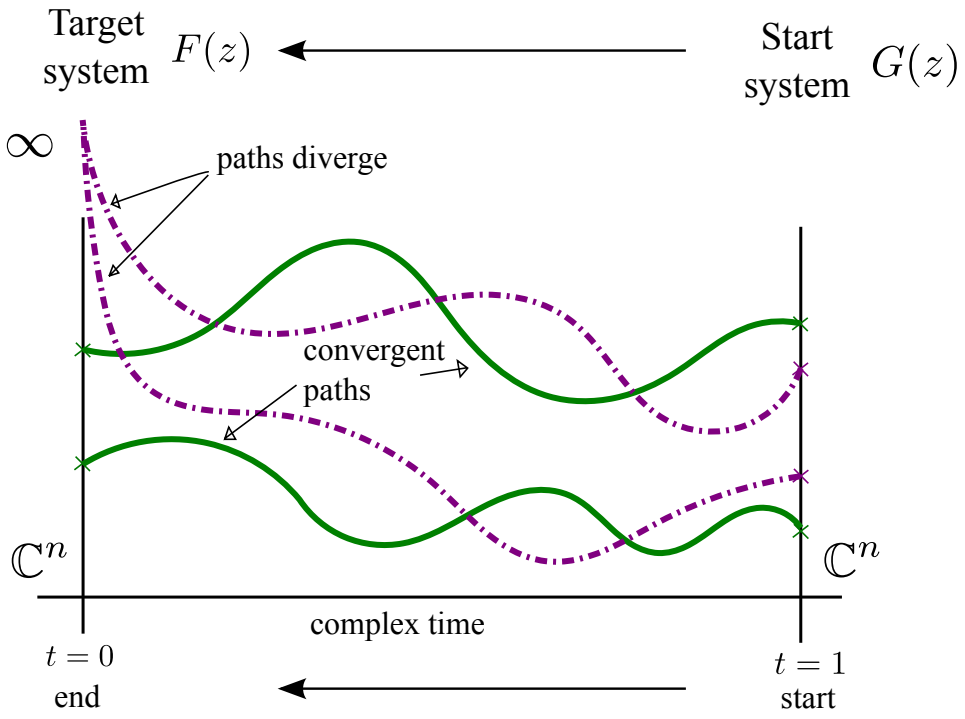


FIGURE 2.1. Generic homotopy continuation.

generic in order to implicitly use the so-called γ-trick [10]. We call $H(z;t)$ the *standard γ-trick homotopy*. Without γ, we call this the *standard homotopy*.

2.2.1. COMPLEX TO REAL EXAMPLE. Using homotopy methods, one must track paths through complex space in order to guarantee finding all the zero-dimensional solutions. These paths must be tracked in complex space, as some some solutions of the start system that are real track to complex solutions of the target system and vice-versa. Similarly, complex solutions can track to complex solutions, and real solutions can homotope to real solutions. This is illustrated by the following example.

Consider a target system

$$f_1(x_1, x_2) = x_1^2 x_2 - 2x_2 - 1$$

$$f_2(x_1, x_2) = x_1 x_2 - 3x_1 - 2$$

and a start system

$$\begin{aligned} g_1(x_1, x_2) &= x_1^3 - 1 \\ g_2(x_1, x_2) &= x_2^2 + 1 \end{aligned}$$

and the standard γ-trick homotopy, i.e., $H(z;t) = \gamma t g(z) + (1-t)f(z)$.

With γ being any random complex number, then all 6 of the solutions of the start system are complex. Using `bertini`, we classify 2 of the target solutions as being real.

2.2.2. γ-TRICK. The γ-trick is a quintessential tool when performing homotopy continuation. Take for instance the previous homotopy example without the random γ. If the homotopy were to be tracked in this way, it would encounter a point at some time t_0 where the solutions were singular. In other words at t_0, two paths come together, and a solution of multiplicity of 2 occurs. After we pass the time t_0 when this singularity happens, we are not guaranteed of correctly tracking both paths to the target system of interest, as 2 paths would have degenerated into 1 and we would end up following only 1 path. Fortunately, this is a probability 0 event, and we can ignore this fatality in practice. Indeed for this example, when performing a user-defined homotopy using `bertini`, we obtain 5 solutions, with 1 of multiplicity 2, contrary to our previous results. The γ-trick is designed to avoid such issues. By default, `bertini` uses the γ-trick when solving a system with total degree homotopy.

2.2.3. BOUNDS ON START SYSTEMS. Let $d_i = \deg(g_i)$, where g_i is a polynomial of our target system. The corresponding polynomial of the total degree start system is then $f_i = x_i^{d_i} - \gamma_i$, where γ_i is a random complex value. Essentially, we consider the d_i'th roots

that lie on a circle of radius $\sqrt{\gamma_i}$ built in a combinatorial manner. The total degree start system has the largest number of paths that we can track in a homotopy.

The choice of a start system can greatly impact the computational run time of tracking to the target system. A bound exists on the number of paths for any start system, namely the total degree, or the Bézout count, which is

$$\Pi_i deg(f_i).$$

The number of paths of a start system is of particular importance when one wants to solve the same system repetitively, but with slightly different coefficient values. Constructing a total degree start system every time for each coefficient value of interest becomes computationally prohibitive as many of the paths are extraneous, and, in fact, if the same start system is used and the coefficient values slightly change, the same paths will track off to infinity, as will be made more clear in the following section. If the number of paths that are extraneous greatly exceeds the number of paths that are finite, it becomes advantageous to construct start systems that reduce the number of paths. M-homogeneous [58] and [81], linear product [83] and [75], and polyhedral [46] and [50], [37][78], and [80] methods exist to construct start systems besides the total degree start system.

Depending on the system at hand, the computational resources required to attain the solutions of the different types of start systems varies. One very special type of start system, called the *parameter homotopy start system*, is extremely useful. This type of start system is used in particular situations when the number of solutions is significantly less than the total degree and when we want to solve the same system repetitively, but with different coefficient values. With parameter homotopy start systems, one does an initial run, letting all of the

coefficients be random points in $\mathbb{C}$. With probability 1, the solution space of this generic system will have the same geometry everywhere, i.e. the number of finite, nonsingular, complex solutions will be the same for all choices of parameters away from a measure 0 set. One then uses these solutions of the generic start system to track to the (possibly multiple) target systems of interest where only the coefficients have changed [61]. Parameter homotopies will be discussed in more detail in §3.3.

2.2.4. Software. Several homotopy continuation software packages have been developed: `bertini` [13], Phcpack [77], HOM4PS2.0 [50], POLYSYS_PLP [83], POLYSYS_GLP [75], HOMPACK [82], and PHoM [37] throughout the years.

Besides continuation software packages, a handful of software packages handle symbolic computations and are known as Computer Algebra Systems (CAS). These programs are also widely used in practice. Examples of these include Maple, Mathematica, MatLab (via the symbolic tool package), SymPy (a Python-based CAS), Macaulay2, Singular, Magma, and GAP. Each of these programs has its own strengths and weaknesses, as some are more general purpose (i.e. Mathematica, Maple, and MatLab), while others are written for very specific types of computations (Macaulay2, Singular, MAGMA, and GAP). This dissertation will cover numerical or hybrid symbolic-numerical algorithms (see §4 and §7) and as such will not discuss these other software packages or their capabilities, beyond mentioning that they belong in the wide category of computational algebraic geometry software. (There are many other mathematical packages than those listed.)

2.3. Varieties and Witness Sets

Let $I \subset \mathbb{C}[x_1, x_2, \ldots, x_n]$ be an ideal and $f_1, f_2, \ldots, f_m : \mathbb{C}^n \mapsto \mathbb{C}$ be generators of I.

Classically, the variety of I, denoted as $V(I)$, is $V(I) = \{p \in \mathbb{C}^n \mid f_1(p) = f_2(p) = \cdots = f_m(p) = 0\}$.

$V(I)$ is *irreducible* if it cannot be written as a the union of two proper algebraic subsets of $V(I)$.

Then,

$$V(f) = \bigcup_{i=0}^{D} \bigcup_{j \in \Gamma_i} V_{i,j},$$

where the irreducible components $V_{i,j}$ are indexed by dimension (up to $D = \dim Z$) and within each dimension i by a finite indexing set Γ_i. The above statement means that each variety can be decomposed into irreducible components of different degrees and dimensions.

A numerical irreducible component, $V_{i,j}$ is then represented as an object with three main pieces of information:

(1) $f = \{f_1, f_2, \ldots, f_m\}$, i.e. the generators of the ideal represented as polynomials;

(2) a linear subspace, $L_{i,j}$ where $\dim(L_{i,j}) = \operatorname{codim}(V_{i,j})$; and

(3) a set of witness points, $W_{i,j} = \{p \in \mathbb{C}^n \mid l_i \in L \Rightarrow l_i(p) < \epsilon \text{ and } f_i(p) < \epsilon\}$ with ϵ as some user-defined tolerance. Each p is chosen as the approximation of a sequence of points that eventually converges by testing when two consecutive points in the generated sequence are within ϵ.

$V_{i,j} = \{f, L_{i,j}, W_{i,j}\}$ is known as a "witness set," as it encapsulates the information in order to "witness" an irreducible component. $\mid W_{i,j} \mid$ gives the degree of the component. Also, $V_{i,j}$ is uniquely determined by the choice of $L_{i,j}$ and the corresponding $W_{i,j}$ that are on the component. It is possible, however, to have two irreducible components of the same or

different dimension and of the same or different degrees. During the steps of computing a witness set, no distinction is made as to which points belong to which components, other than to know the dimension of the component to which the witness points belong. A pair of other computations, called monodromy and the trace test, is then performed in order to group the points together with the correct component [72]. Phcpack [77] and `bertini` [13] can compute the numerical irreducible decompositions of a variety. Phcpack allows users to choose this option via a menu-based system, while `bertini` is implemented with a user-defined settings that are read by from the main program. One could also create a redirect input file in order to use Phcpack without the menu.

2.3.1. Positive Dimensional Examples. For most systems, scientists are only interested in real solutions. For quite awhile, little was known about how to find real solutions in complex components. Recent progress has been made in this regard, see [54] and [15]. Apart from that, isolated real solutions (isolated over $\mathbb{C}$) were the only real solutions that could be found quite readily. These isolated points correspond to zero-dimensional varieties. However, positive dimensional varieties are interesting in their own right. We provide two examples of systems with positive dimensional varieties and classify their components.

2.3.1.1. *Example 1: The Twisted Cubic and a Sphere.* The twisted cubic is one of the simplest examples of a variety that is neither a hypersurface nor linear. It is an irreducible curve of degree 3. Its solution set satisfies the following polynomial system:

$$\begin{aligned} T_1(x_1, x_2, x_3) &= (x_1 x_3 - x_2^2) \\ T_2(x_1, x_2, x_3) &= (x_2 - x_3^2) \\ T_3(x_1, x_2, x_3) &= (x_1 - x_2 x_3). \end{aligned}$$

We can union the twisted cubic with that of a hypersurface, in this case the complex unit 3-sphere. The system of equations for this system is

$$\begin{aligned} f_1(x_1,x_2,x_3) &= (x_1x_3 - x_2^2)(x_1^2+x_2^2+x_3^2-1) \\ f_2(x_1,x_2,x_3) &= (x_2 - x_3^2)(x_1^2+x_2^2+x_3^2-1) \\ f_3(x_1,x_2,x_3) &= (x_1 - x_2x_3)(x_1^2+x_2^2+x_3^2-1) \end{aligned}$$

We know that we have two irreducible components, one of dimension 2 (the sphere) and one of dimension 1 (the twisted cubic). `bertini` performs a numerical irreducible decomposition and verifies this to be the case.

2.3.1.2. *Example 2.* We can also consider varieties that are generated by a system of equations that we do not understand *a priori*.

Let

$$\begin{aligned} f_1(x_1,x_2,x_3) &= ((x_1 - \frac{1}{2})^2+x_2^2+x_3^2-1)(x_1x_2x_3 + x_2 - 2x_3 + 5) \\ f_2(x_1,x_2,x_3) &= ((x_1 + \frac{1}{2})^2+x_2^2+x_3^2-1)(x_2^2x_3 + 5x_1 - 3x_2 - 4) \end{aligned}$$

By doing a positive-dimensional `bertini` run, we are able to find the decomposition of the solutions of this system. According to `bertini`, a total of 4 components exist of codimension 2. There is one component each of degree 2 and 5 and two components of degree 6. These results make sense when we consider the system in the following way.

If we let $a_1(x_1,x_2,x_3) = (x_1^2 - \frac{1}{2})^2 + x_2^2 + x_3^2 - 1$, $a_2(x_1,x_2,x_2) = x_1x_2x_3 + x_2 - 2x_3 + 5$, $a_3(x_1,x_2,x_3) = (x_1 + \frac{1}{2})^2 + x_2^2 + x_3^2 - 1$, and $a_4(x_1,x_2,x_3) = x_2^2x_3 + 5x_1 - 3x_2 - 4$, then we see we have a total of 4 combinations that define these curves, namely $< a_1, a_3 >$, $< a_1, a_4 >$,

$< a_2, a_3 >$, and $< a_2, a_4 >$. When looking at the variety generated by $< a_1, a_3 >$, we notice that the maximal degree of any monomial term in either of the two generators is of degree 2. In addition, we have that at least one monomial consists of only a single variable and is of degree 2. Thus, we have that $< a_1, a_3 >$ yields a component that is codimension 2 and is of degree 2. Now, at this point we see that we have two different spheres (i.e. a_1 and a_3), both of which generate two different ideals when intersected with a degree 3 hypersurface, i.e. $< a_1, a_4 >$ and $< a_2, a_3 >$ (these two intersections are now of codimension 2), and we find the variety of these two ideals both have degree 6, as computed by `bertini`. This leaves us with the last component generated by $< a_2, a_4 >$ which is of degree 5. We should make a note that these results for this example could have been done using Phcpack [77]. Unless otherwise noted, we will make use of programs that utilize the `bertini` library.

CHAPTER 3

Topics

This chapter will provide introductory material on the rest of this dissertation. §3.1 details decoupling, a new algorithm that applies to certain polynomial systems with specific structures. §3.2 discusses a completely real-tracking algorithm, and heuristics that have been employed in a new software package `galeDuality`. §3.3 considers parameter homotopies and briefly introduces another software package, `paramotopy`. §3.4 outlines applications of parameter homotopies using two examples coming from biochemical reaction networks and rock magnetism. Finally, §3.5 describes an algorithm that makes use of the monodromy action in order to avoid high precision [11].

3.1. Decoupling

In this section, we propose an efficient method for solving polynomial systems with a particular structure. This structure is very specific but arises naturally when computing the critical points of a symmetric polynomial energy function. This novel numerical solution method is based on homotopy continuation and may apply more broadly than this specific setting. An illustrative example from magnetism is presented in §6.3.2.

Let $f : \mathbb{C}^{Nk} \to \mathbb{C}^{Nk}$ be a polynomial system of the form

$$f(z_1, z_2, \ldots, z_N) = \begin{cases} f_1(z_1, z_2, \ldots, z_N) \\ f_2(z_1, z_2, \ldots, z_N) \\ \vdots \\ f_N(z_1, z_2, \ldots, z_N) \end{cases}$$

$$(1) \qquad = \begin{cases} g_1(z_1) & & & + & h_1(z_1, z_2, \ldots, z_N) \\ & g_2(z_2) & & + & h_2(z_1, z_2, \ldots, z_N) \\ & & \ddots & \vdots & \\ & & g_N(z_N) & + & h_N(z_1, z_2, \ldots, z_N), \end{cases}$$

where each f_i, g_i, and h_i is a k-tuple of polynomials and the z_i are non-overlapping k-tuples of variables for $i = 1, \ldots, N$. Notice that, for each i, g_i depends only on variables z_i while h_i may depend on all of the variables. We further require the polynomials g_i to all have the same monomial structure. As a very simple example, suppose that the set of equations breaks into 1-tuples and, for each i, g_i is a single polynomial given by $g_i(x) = a_i x_i^3 + b_i$, with the coefficients a_i and b_i being complex numbers. Then, the only difference between $g_i(x_i)$ and another block $g_j(x_j)$ is in their coefficients and the labels on their variables.

The isolated solutions of

$$f(z_1, z_2, \cdots, z_N) = 0$$

in $\mathbb{C}^{N_k}$ (including all real solutions isolated over the complex numbers) may be approximated to arbitrarily high accuracy with numerical algebraic geometry, based on homotopy continuation. In this section, we present a novel method, *decoupling*, that will provide at least some of these solutions (sometimes all) much more efficiently and in a more scalable manner. The situations in which we can find all solutions have to adhere to certain criteria as outlined by Canny and Rojas [78].

While the structure required in (1) may seem restrictive, it is precisely the structure attained when computing the critical points of a symmetric polynomial energy function by solving the system of first partial derivatives. We encountered this structure when working

on a particular set of magnetism problems with multiple dipoles. The terms $g_i(z_i)$ come from the energy that dipole i would have in the absence of the other dipoles, while the terms $h_i(z_1, z_2, \cdots, z_N)$ come from interactions between dipoles. $g_i(z_i)$ come from self-interactions of each dipole while the *interacting terms* $h_i(z_1, z_2, \cdots, z_N)$ come from interactions between different dipoles. The scenario is not restricted to magnetism and should arise whenever there is a system with both long-range interactions and internal energies. In fact, polynomial systems of this sort could come from any number of applications or constructions, so this method is not restricted to magnetism or perhaps even to physics.

3.1.1. Decoupling Algorithm.

Algorithm 1: Decoupling

Input: Polynomial system $f : \mathbb{C}^{Nk} \to \mathbb{C}^{Nk}$, broken into k-tuples g_i and h_i for $i = 1, \ldots, N$, as in (1). Let $z_1, \ldots, z_N$ be N k-tuples of variables.

Output: Solutions of $f = 0$.

1. Solve $g_*(z) = 0$, a $k \times k$ polynomial system with the same monomial structure as each $g_i(z_i)$ but random, complex coefficients. Use any standard homotopy continuation technique for this.
2. For each $i = 1, \ldots, N$, use a parameter homotopy from $g_*(z)$ to find all isolated solutions of $g_i(z_i) = 0$ in $\mathbb{C}^k$.
3. Combinatorially concatenate all solutions from all blocks to form $N-tuples$ of solutions $(s_1, \ldots, s_N)$ for $g(z_1, z_2, \ldots, z_N) = 0$, where g is defined in (1). These are the start solutions S for the final step.
4. Solve the original problem $f(z_1, z_2, \ldots, z_N)$ by way of a parameter homotopy from $g(z_1, z_2, \ldots, z_N)$ to $f(z_1, z_2, \ldots, z_N)$, starting from all the solutions in S.

We make a note that we can use a variation of *decoupling* if each subsystem does not have the same monomial structure in all of the $g_i(z_i)$ provided that both of the following are true:

1) each generic subsystem have as the number of complex solutions the total degree of that subsystem; and
2) the $\deg(g_i(z_i)) > \deg(h_i(z_1, z_2, \ldots, z_n))$.

The only portion of the algorithm that is different between the variant of decoupling and the original is that we do not solve for a generic $g_i(z_*)$ (due to the removal of the restriction of having the same monomial structure), but rather solve for generic $g_{i_j}(z_*)$, where each g_{i_j} is one representative of a group of subsystems with the same monomial structure.

We provide two systems that utilize the concepts behind *decoupling* and this variant. In §6.3.2, we show one system that has a physical application to rock magnetism and in §3.1.2 a scenario where it is not necessarily true to attain all the complex solutions when one of the subsystems does not have the total degree number of solutions. We see that decoupling does not fail so much as the system to which we attempt to employ the algorithm does not have the necessary structure.

In the rock magnetism application, the subsystems have solutions that diverge to infinity, however, all solutions of the original polynomial system are found in the end. This example provides evidence that for certain highly structured systems, *decoupling* can still work, even when the system of interest does not adhere to our exact criteria. Indeed, one could potentially use techniques similar to Canny and Rojas [21].

3.1.2. Decoupling Fails to Capture All Solutions. Suppose

$$\begin{aligned} f_1 &= p_1x^3y + p_2y^2x^2 + p_3y^2x + p_4x^2y - 1 + p_{12}wz + p_{13}wz^2y \\ f_2 &= p_5x^2y + p_6y^2x - 4 + p_{14}w^2y + p_{15}wxz \\ f_3 &= p_7wz + p_8z^3 + p_9w - 3 + p_{16}yz + p_{17}x^2 \\ f_4 &= p_{10}wz - 1 + p_{18}xz + p_{19}y^2x \end{aligned}$$

is the system we are interested in solving.

If we naïvely choose our two subsystems to be

$$\begin{aligned} f_1 &= p_1x^3y + p_2y^2x^2 + p3y^2x + p4x^2y - 1 \\ f_2 &= p_5x^2y + p_6y^2x - 4 \end{aligned}$$

and

$$\begin{aligned} f_3 &= p_7wz + p_8z^3 + p_9w - 3 \\ f_4 &= p_{10}wz + p_{11}w^3 - 1 \end{aligned}$$

we will not capture all the solutions if we employ the variant of decoupling.

The first subsystem does not have the Bezòut count as the total number of systems–it has only 5 instead of 12–but the second subsystem does. This is a violation of the first criteria listed above. When combinatorially building the start system when we bring in the coupled terms between the subsystems, we only have $5(9) = 45$ starting solutions. Out of

these 45 starting solutions, 30 of the paths remain finite, and the other paths diverge. The system that we are attempting to solve has 57 solutions.

We are able to attain 30 of the 57 solutions in approximately .208 seconds using decoupling and regeneration when solving the individual subsystems and about .202 seconds using decoupling and no regeneration when solving the individual blocks. On the other hand, while we were able to find all the solutions using `bertini` with regeneration, the time it took to solve the system, on average, was 1.698 seconds. In addition, the average time it took to solve the system with using total degree homotopies in `bertini` was .879 seconds.

We note that decoupling and its variant are both highly useful when a large number of subsystems have the same monomial structure (up to a renaming of the variables in the subsystems).

3.2. Completely Real-Tracking Algorithms

As noted earlier (see §2.2.1), the main problem in tracking real curves lies in the inability to guarantee that one will find all isolated nonsingular solutions. However, an algorithm does exist for finding all the real solutions by tracking only over the reals. This algorithm, called Khovanskii-Rolle Continuation, developed by Bates and Sottile [12] is a hybrid symbolic-numeric algorithm that tracks completely real curves using arclength continuation and is capable of finding all the real solutions of a polynomial system without finding all of the complex solutions first. This is contrary to the case when solving a system with homotopy continuation. In particular, Khovanskii-Rolle is well suited for *fewnomial* systems. A fewnomial system is a polynomial system where the total number of monomials in the system is only "few" more than the number of variables. The theory of fewnomials has grown with much development done by Khovanskii [49]. Fewnomial theory has recently been used to

give a bound for the number of real solutions of polynomial systems [17] and [3]. §4.1 describes the symbolic gale transform of a fewnomial system and discusses techniques on how to precondition this transform so as to make the Khovanskii-Rolle Continuation algorithm well-conditioned numerically. The symbolic gale transform algorithm is implemented in a new software package called `galeDuality`.

3.3. Parameter Homotopies

Numerical algebraic geometry provides a number of efficient tools for approximating the solutions of polynomial systems. One such tool is the parameter homotopy, which can be an extremely efficient method to solve several polynomial systems that differ only in coefficients. This technique is useful for solving a parameterized family of polynomial systems at multiple parameter values.

Parameter homotopy continuation uses a slight modification of normal homotopy continuation methods. An initial standard homotopy continuation run is conducted at a generic point in parameter space. Theory guarantees that this initial target system has the maximum number of nonsingular complex solutions [53, 61]. One then performs another homotopy, called a *parameter homotopy*, by continuously deforming the parameters, or coefficients, of the polynomial system from the generic parameter values to the parameter values of interest.

Suppose $f : \mathbb{C}^n \times \mathbb{C}^m$, where n is the number of variables and m is the number of parameters. Suppose $z \in \mathbb{C}^n$ and $q \in \mathbb{C}^m$, and consider a system $f(z, q)$ that is linear in the parameter space. Let q^* be a generic point in our parameter space. Once we solve the system at this generic point, we can use this general system as our new start system. For a particular point in parameter space of interest, q_i, we construct a parameter homotopy of

the following form:

$$\begin{aligned} H(z,t) &= tf(z;q) + (1-t)f(z;q_i) \\ &= f(z;qt) + f(z;q_i(1-t)), \\ &= f(z;qt+(1-t)q). \end{aligned}$$

The power behind parameter homotopies lies in the fact that in most systems that arise from applications, the number of finite solutions is is *deficient* in comparison with the Bezòut count. We will see two examples of this reduction in the number paths in §5.6.

Parameter homotopies have recently been used in several areas of application and have been implemented in at least two software packages, Phcpack [77] and `bertini` [13]. A module of bertini, `paramotopy`, is an optimized parallel implementation of this technique, making use of the `bertini` library. The novel features of this program unavailable elsewhere include allowing for computing the simultaneous solutions of arbitrary polynomial systems in a parameterized family on an automatically generated (or a manually provided) mesh in the parameter space of coefficients, front ends and back ends that are easily specialized to particular classes of problems, lookup tables for solutions to particular parameter points, and adaptive techniques for solving polynomial systems near singular points in the parameter space. This last feature automates and simplifies a task that is important but often misunderstood by non-experts.

`paramotopy` [4] is being developed jointly by Daniel Bates, Daniel Brake, and the author of this dissertation. `paramotopy` will be discussed in more detail in §5.2, and we will explore

ways to use the software in areas of scientific inquiry to answer questions about certain problems.

3.4. Applications of Parameter Homotopies

Parameter homotopies are not new and have been used in several areas of applications [20, 41, 69]. In addition, parameter homotopies have also been found to have applications in biochemical reaction networks [25], epidemiology models [62], statistical mechanics [18], rock magnetism [64] and in certain areas of phenomenological string theory [55].

We will show how the `paramotopy` package can and has been used in two examples: biochemical reaction networks and rock magnetism in §6.

3.5. Using Monodromy to Avoid Adaptive Multiprecision

The current implementation of `bertini` allows for the use of adaptive multiprecision techniques during path-tracking when the condition number becomes too large [9]. This feature is useful since the number of digits of accuracy to track a path near a singularity increases. However, it is not necessary to store all those digits while tracking the entire path. Higher precision correlates to increased computational time. An alternative is to monitor the condition number of the Jacobian and construct an ad-hoc homotopy in order to avoid the use of high precision. In the final chapter, we will consider an example where a user-defined homotopy is given and adaptive multiprecision reaches the maximum precision level allowed, given by the maximum settings, and the path fails. It is important to note that this pathological example is designed purposely in such a manner that two of the paths meet at a branch point, which is why `bertini` fails to find all the solutions. When utilizing the so-called "gamma trick", with probability 1, paths are generic and paths will not meet

at a branch point (but the "gamma trick" is ignored in this user-defined homotopy). The purpose of this algorithm is to show that in some situations, the random γ chosen may bring the path too close to an ill-conditioned region and, as such, provides a manner in which to avoid the poor numerics that occur. This algorithm will be discussed more in §7.

CHAPTER 4
Gale Duality

Given a polynomial system with very few monomials, one could use standard homotopy methods to approximate all real, isolated solutions. However, this could be wasteful, particularly if very few real solutions exist relative to the number of complex solutions. An alternative technique which *only* finds real solutions and ignores complex solutions is the Khovanskii-Rolle continuation algorithm [12].

The Khovanskii-Rolle continuation algorithm utilizes a generalization of Rolle's Theorem by Khovanskii. Instead of solving the original fewnomial system, the algorithm solves an equivalent system, called the gale dual system. Sometimes, we will refer to the gale dual system as the gale system or more simply as the dual. This transformation, called the gale transform, from a fewnomial system to the gale system relies on a scheme-theoretic isomorphism between the solutions of the master functions that lie in the positive chamber of the complement of a particular hyperplane arrangement and the isolated real solutions that lie in the positive orthant of the fewnomial system [16]. This technique can be used to find all real solutions by changing which chamber is the positive one. This corresponds to multiplying linears that define the boundaries of the chamber by -1. It is important to note that one does not necessarily have to restrict to isolated solutions as the transformation still works for positive dimensional components, but implementation becomes an issue and so the positive dimensional case is beyond the scope of this dissertation. The following section describes the algorithm for converting a fewnomial system to its corresponding dual via the gale transform.

4.1. Gale Transform with Khovanskii-Rolle

Let $F : \mathbb{R}^n \mapsto \mathbb{R}^n$ be a fewnomial system with $n + \ell + 1$ monomials in the support of F. The requirement for the number of monomials to be $n+\ell+1$ as opposed to $n+\ell$ allows us to include the constant monomial in the support of F and simpler notation naturally follows. Let $G : \mathbb{R}^\ell \mapsto \mathbb{R}^\ell$ be the corresponding polynomial system of master functions that we will construct. Notice that a $10000 \times\ 10000$ polynomial system with 10003 total monomials will be transformed into a $2 \times\ 2$ polynomial system; this is the real power behind this method.

Let $\{m_1, \ldots, m_{n+\ell}, 1\}$ be the monomials in the support of F. Consider the exponent matrix of the monomials without the column that corresponds to the constant monomial. Call this matrix E. More precisely, the ij^{th} entry in E is the exponent of the i^{th} variable in the j^{th} monomial. The key to the gale transform is in constructing vectors that span the null space of the support of the monomials.

Let C be the coefficient matrix of the fewnomial system F with the $n + \ell + 1$ column representing the coefficients for the constant monomial and the rest of the columns of E and C to match up with the corresponding monomials, i.e. the i^{th} column of E and C represent the exponents of the i^{th} monomial for one of the variables and the coefficient of the i^{th} monomial for one of the equations, respectively. The choice of the order of the monomials in these matrices does not make any significant difference in gale transform algebraically, except for some choices as will be shown in the example found in §4.2 and §4.3, but does affect the numerics and computational time of the Khovanskii-Rolle algorithm (see §4.3.2).

By performing gaussian elimination on the coefficient matrix, n of the monomials can be parametrized in terms of the other $\ell + 1$ monomials. By letting $s_1, \ldots, s_\ell$ be new variables, each of the other n monomials can be written as linear expressions in terms of $s_1, \ldots, s_\ell$.

Let $L_i(s_1, \ldots, s_\ell)$ be these linear expressions for $1 \leq i \leq n + \ell$. Let $\mathcal{B} = \{b_1, \ldots, b_\ell\}$ be a basis for the null space of E. Note that each b_i lives in $\mathbb{Z}^{n+\ell}$. Once more, a choice in the algorithm, in this case the choice of the basis vectors that span the null space, does not change the symbolic nature gale transform but may affect numerical computations during Khovanskii-Rolle. We see then that an infinite number of choices exist for the symbolic gale transform, but only a handful of these choices are "good." The key behind the construction

Algorithm 2: Transforming fewnomials F into master functions G

Input: ℓ, n, $supp(F)$ as an $n \times (\ell + n)$ matrix E,and the coefficient matrix C as an $n \times (\ell + n + 1)$ matrix. **Output**: A system G of ℓ fewnomials in ℓ variables, gale dual to F.

1) Choose an order for the monomials and reorder the columns of C.
2) Reduce C to reduced row echelon form.
3) From the reduced C, record parameterizations $L_i(s_1, \ldots, s_\ell)$ for each monomial.
4) Compute ℓ generators of the null space N of E such that each coordinate is an integer.
5) Use the generators in N to create the dual gale system of ℓ master functions.

of the gale transform is using some basis for the null space of the exponent matrix. By raising each monomial to the associated power in the corresponding coordinate of the basis vector and then multiplying these all together, a new system arises that is equal to 1, since, after simplification, each of the variables is raised to the 0^{th} power.

Thus,

$$\begin{aligned} G_1 &:= m_1^{b_{1,1}} m_2^{b_{1,2}} \dots m_{n+\ell}^{b_{1,n+\ell}} = 1 \\ G_2 &:= m_1^{b_{2,1}} m_2^{b_{2,2}} \dots m_{n+\ell}^{b_{2,n+\ell}} = 1 \\ &\vdots \\ G_\ell &:= m_1^{b_{\ell,1}} m_2^{b_{\ell,2}} \dots m_{n+\ell}^{b_{\ell,n+\ell}} = 1 \end{aligned}$$

become the corresponding *master functions* of the gale dual of the fewnomial system.

Since each component in b_i need not be positive, the master functions are rational functions. Separating the positive and negative exponents into two distinct sets for each b_i, say b_i^+ and b_i^- with n_1 and n_2 being the size of the two sets respectively, the following notation is obtained to represent the master functions:

$$\begin{aligned} G_1 &:= \prod_{1 \le k_+ \le n_1} m_{k_+}^{b_{1,k_+}} - \prod_{1 \le k \le n_2} m_{k_-}^{b_{1,k_-}} = 0 \\ G_2 &:= \prod_{1 \le k_+ \le n_1} m_{k_+}^{b_{2,k_+}} - \prod_{1 \le k \le n_2} m_{k_-}^{b_{2,k_-}} = 0 \\ &\vdots \\ G_\ell &:= \prod_{1 \le k_+ \le n_1} m_{k_+}^{b_{1,k_+}} - \prod_{1 \le k \le n_2} m_{k_-}^{b_{\ell,k_-}} = 0 \end{aligned}$$

with m_{k_+} and m_{k_-} being the appropriate monomials that correspond to the exponent of b_{i,k_+} and b_{i,k_-} for $1 \le i \le \ell$.

The next step in the gale transform is to substitute $L_i(s_1, \ldots, s_\ell)$ for m_i or L_{k_+} and L_{k_-} for the appropriate m_{k_+} and m_{k_-} in the master functions. Thus,

$$\begin{aligned} G_1 &:= \prod_{1 \le i \le n+\ell} L_i^{b_{1,i}} = 1 \\ G_2 &:= \prod_{1 \le i \le n+\ell} L_i^{b_{2,i}} = 1 \\ &\vdots \\ G_\ell &:= \prod_{1 \le i \le n+\ell} L_i^{b_{\ell,i}} = 1 \end{aligned}$$

are the master functions after the appropriate substitutions.

After clearing denominators, the Khovanskii-Rolle algorithm takes over at this point and returns solutions in terms of the new variables $s_1, \ldots, s_\ell$. An "unwrapping" algorithm is applied to convert the isolated real solutions of the master functions to their corresponding isolated real solutions of the original polynomial system as described in Algorithm 3.

Algorithm 3: Transforming master function solutions to fewnomial solutions

Input: All input to Algorithm 2 and the set $S = \{s_i : i = 1, ..., N\}$ of solutions of the gale dual system of master functions in the positive orthant.

Output: The set $T = \{t_i : i = 1, ..., N\}$ of positive real solutions of the original fewnomial system F.

for i := 1 to N **do**

1) Evaluate the linear functions at s_i to form the vector v.

2) Choose n columns of E to form the matrix E'.

3) Solve the linear equation $E' \cdot x = v$ for x.

4) Exponentiate the entries of x to form t_i.

end for

4.2. Example of Algorithm 2.

Consider the (Laurent) polynomial system

$$
\begin{aligned}
a^{-1}b^2c^2d - \left(\tfrac{1}{2}b^2c + 2b^{-4}c^{-7}d^{-5}e^{-1} - 1\right) &= 0,\\
ac - \tfrac{1}{2}(\tfrac{1}{4}b^2c - b^{-4}c^{-7}d^{-5}e^{-1} + 1) &= 0,\\
bc^4d^4 - \tfrac{1}{4}(-\tfrac{1}{4}b^2c - 3b^{-4}c^{-7}d^{-5}e^{-1} + 6) &= 0,\\
d - \tfrac{1}{2}(-\tfrac{1}{4}3b^2c - 2b^{-4}c^{-7}d^{-5}e^{-1} + 8) &= 0,\\
e - (-\tfrac{1}{2}b^2c + 2b^{-4}c^{-7}d^{-5}e^{-1} + 3) &= 0,
\end{aligned}
\tag{2}
$$

with support $\{a^{-1}b^2c^2d, ac, bc^4d^4, d, e, b^2c, b^{-4}c^{-7}d^{-5}e^{-1}, 1\}$, i.e., exponent matrix

$$
E = \begin{bmatrix}
-1 & 1 & 0 & 0 & 0 & 0 & 0\\
2 & 0 & 1 & 0 & 0 & 2 & -4\\
2 & 1 & 4 & 0 & 0 & 1 & -7\\
1 & 0 & 4 & 1 & 0 & 0 & -5\\
0 & 0 & 0 & 0 & 1 & 0 & -1
\end{bmatrix}
$$

and coefficient matrix

$$
C = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & -1/2 & -2 & 1\\
0 & 1 & 0 & 0 & 0 & -1/8 & 1/2 & -1/2\\
0 & 0 & 1 & 0 & 0 & 1/16 & 3/4 & -3/2\\
0 & 0 & 0 & 1 & 0 & 3/8 & 1 & -4\\
0 & 0 & 0 & 0 & 1 & 1/2 & -2 & -3
\end{bmatrix},
$$

where the first seven columns correspond to the seven nonconstant monomials in the same order as above and the final column corresponds to the constant.

For simplicity, this fewnomial system is presented with a coefficient matrix in reduced row echelon form for a particular choice of parameters (b^2c and $b^{-4}c^{-7}d^{-5}e^{-1}$), so Steps 1 and 2 of Algorithm 1 are complete. Renaming these two monomials s_1 and s_2, the monomial ac is parametrized, for this example, so that $ac := \frac{1}{8}s_1 - \frac{1}{2}s_2 + \frac{1}{2}$. This, and similar expressions for the other monomials, result in the following seven linear functions:

$$
\begin{aligned}
L_1(s_1, s_2) &= \tfrac{1}{2}s_1 + 2s_2 - 1 \\
L_2(s_1, s_2) &= \tfrac{1}{8}s_1 - \tfrac{1}{2}s_2 + \tfrac{1}{2} \\
L_3(s_1, s_2) &= -\tfrac{1}{16}s_1 - \tfrac{3}{4}s_2 + \tfrac{3}{2} \\
L_4(s_1, s_2) &= -\tfrac{3}{8}s_1 - s_2 + 4 \\
L_5(s_1, s_2) &= -\tfrac{1}{2}s_1 + 2s_2 + 3 \\
L_6(s_1, s_2) &= s_1 \\
L_7(s_1, s_2) &= s_2
\end{aligned}
\tag{3}
$$

Now, given a vector $b_j = [b_{j,1}, \ldots, b_{j,7}]$ in the nullspace of E, $\prod_{i=1}^{7} m_i^{b_{j,i}} = 1$, where m_i is the i^{th} monomial in the support. Substituting in the linear equations of the previous paragraph, $\prod_{i=1}^{7} L_i(s_1, s_2)^{b_{j,i}} = 1$. For example, one basis for the nullspace of E is

$$\{[-1, -1, 2, -2, 1, 2, 1]^T, [4, 4, 2, 3, 3, 1, 3]^T\}.$$

Combining these with the linear functions of 3 results in the system of master functions

$$
\begin{aligned}
G_1(s_1, s_2) &= -2(s_1 + 12s_2 - 24)^2(s_1 - 4s_2 - 6)s_1^2 s_2 - (s_1 + 4s_2 - 2)(s_1 - 4s_2 + 4)(3s_1 + 8s_2 - 32)^2, \\
G_2(s_1, s_2) &= (s_1 + 4s_2 - 2)^4(s_1 - 4s_2 + 4)^4(s_1 + 12s_2 - 24)^2(3s_1 + 8s_2 - 32)^3(s_1 - 4s_2 - 6)^3 s_1 s_2^3 \\
&\quad -68719476736,
\end{aligned}
$$

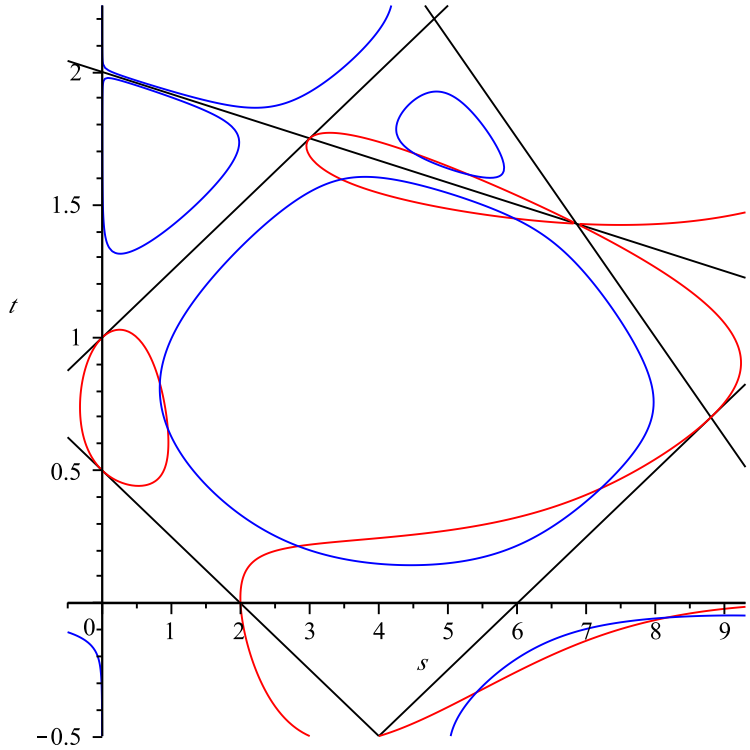


FIGURE 4.1. A plot of $G_1(s_1, s_2)$ and $G_2(s_1, s_2)$ in the complement of the hyperplane arrangement.

by clearing denominators. These functions are defined in the complement of the hyperplane arrangement $L_i(s_1, s_2) = 0$, $i = 1, \ldots, 7$. See Figure 4.1.

4.3. Example of Algorithm 3.

One of the solutions of $G_1(s_1, s_2) = G_2(s_1, s_2) = 0$ in the previous example, found using Khovanskii-Rolle continuation, is

$$s = (5.410516166, -0.335671769).$$

Following the steps of Algorithm 3, we recover one of the solutions of the original fewnomial system.

Evaluating the linear functions at s,

$$v = [1.584633902, -0.652003808, -3.191162879, 1.785323533, 0.370442657].$$

Choosing just the first five columns of M to form M', and solving the linear equation $M' \cdot x = v$, we find

$$x = [-0.801570672, 0.817878335, 0.149566863, -1.151827167, 1.070211793].$$

Exponentiating each entry,

$$t = [0.448623770, 2.265687704, 1.161331119, 0.316058749, 2.915997022].$$

Evaluating t in the five original fewnomials, the Newton residual is less than 10^{-8}, so this is an approximate root of the original system, as expected.

4.3.1. Preconditioning Khovanskii-Rolle. Two main questions arose after the development of the algorithm of transforming a system into its dual gale system: (1) what is a "good" choice of parametrization of the monomials?; and (2) what is a "good" choice of vectors that annihilate the support of the original system? These two questions are natural to consider as any choices made for the parametrization of the monomials and any choices made for the vectors that annihilate the support of the monomials are algebraically equivalent. Even though the gale transformation is completely symbolic in nature, the implementation of Khovanskii-Rolle Continuation relies on numerical computations, and these two choices can dramatically affect the computation of this second algorithm. Why do these two choices affect the speed of the computation?

4.3.1.1. *Preconditioning Parametrization of the Monomials.* The first choice concerns what the resulting polytope that the master functions lie in looks like. Is the polytope "regular"? What type of polytope yields the best run times in the Khovanskii-Rolle continuation algorithm? If the algorithm is tracking a curve near a vertex of the polytope, the curve will have high curvature. The high curvature causes problems in that Newton's method may fail to converge after a prediction.

The linear functions $L_i(s_1, s_2)$ depend on the choice of ℓ monomials to serve as the parameters s_1 and s_2 (in the case $\ell = 2$). Each of the $\binom{n}{2}$ such choices for s_1 and s_2 yields a different positive chamber for the polytope. The curves to be followed are in the interior of the positive chamber, so it is reasonable to expect that the general shape of the polytope could impact the efficiency of the curve-tracing algorithm.

To trace a curve C from some point $p \in C$, we take a small step along the tangent to C, then correct via Newton's method in the orthogonal direction. If, near p, C has high curvature, the predictions along the tangent will not be easily corrected back to C and the step size is halved for a new prediction. Similarly, if Newton's method fails to converge after two iterations, the step size is halved, as with normal homotopy continuation. See Figure 4.2.

Thus, the choice of parametrization of the monomials should be one that yields the most "regular" polytope possible. A heuristic was developed that avoids polytopes with small angles between the supporting linears of the positive orthant, which corresponds to avoiding a polytope with a small area (or volume in higher dimensions). This seems to work well for several examples. Algorithm 4 describes the manner in which a choice for the parametrization of the monomials is to be made.

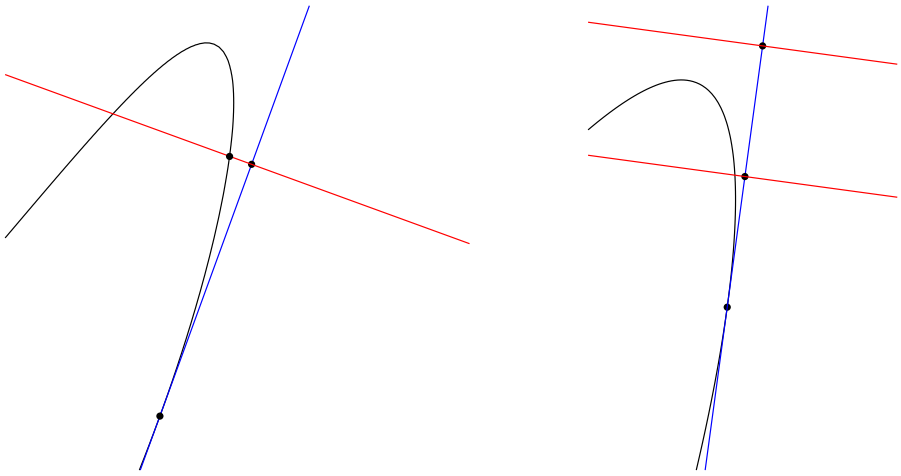


FIGURE 4.2. Correction (along red lines) is feasible in regions of low curvature (left). In the righthand figure, correction is not possible from the initial prediction (along the blue tangent line), so the steplength is halved.

The heuristic used when determining the slicing hyperplane is that the hyperplane should go through the midpoints of the adjacent edges if the angle measures 0 radians. At the other extreme, the hyperplane should pass through the vertex if the angle measures π. For angles between 0 and π, take the obvious, appropriately weighted sum of these two cases.

Once the slicing hyperplane has been chosen, choose a uniform mesh on the portion of the hyperplane passing through the positive chamber. Evaluate the functions corresponding to the curves to be traced at each mesh point. Any sign change indicates that the vertex is approached by the associated curve.

4.3.1.2. *Preconditioning the Annihilating Vectors.* The second question, regarding the choice of basis vectors that annihilate the support, offers more to consider when constructing the gale transform. Although the solutions themselves are not affected by the choice, each curve itself is different depending on the choice of the vector used to annihilate the support. This choice of annihilating vectors is quite different from the choice of the parametrization

Algorithm 4: Choosing parameters when forming master functions

for each possible assignment of monomials to parameters **do**

Find all vertices of the polytope P.

for each vertex v_i **do**

Find both linears supporting v_i.

Slice P with a hyperplane near v_i, to determine whether v_i is approached by the curves to be traced.

Call v_i "active" if it is approached by a curve to be traced.

If v_i is active, take the inner product of the supporting linears and assign that value to v_i.

end for

Find the max and min values assigned to the active vertices and assign the quotient of the max by the min to P.

end for

Select the polytope with value nearest 1.

of the monomials in that the latter does not affect the functions themselves, only the scaling of the polytope.

As noted before, in the corner of each of the chambers of the polytope, the master functions are increasingly singular and instead of tracing the actual master functions themselves, one could, in theory, use monomial curves that approximate the master functions near the corners as the approximations are not nearly as singular as their counterparts. The goal is to then minimize the master function from moving to different chambers of the polytope. How does one do this? As it turns out, this question is related to the choice of basis vectors that annihilate the support. The number of sign changes that occur in each of the basis vectors indicates the number of times that a particular master function will pass through to another

chamber of the polytope [6]. One then wants to compute a subspace with a given basis such that the basis represents the fewest sign changes between coordinates. The scheme of generating all sign patterns is relatively easy and is described in Algorithm 5, and the best can be chosen after all possible representatives have been created.

Algorithm 5: Choosing annihilating vectors

Input: The exponent matrix E of the support of the monomials of F.

Output: A basis for the nullspace of E that has the minimal number of sign changes for the basis elements.

Compute a basis for the nullspace of E and set it to $\mathcal{B}$.

$\ell := \dim(\mathbf{b_i})$ where $\mathbf{b_i}$ in $\mathcal{B}$

$M := \emptyset$

for each $\mathbf{b_i}$ in $\mathcal{B}$ **do**

$m_{b_i} := \max(\gcd(b_{i,j}))$ for $1 \leq j \leq \ell$ and $i \neq j$

$m :=< m_{b_i} >$ for $1 \leq j \leq \ell$

$M := M \cup \{m\}$

end for

$V := \emptyset$

for each m in M **do**

$v := \sum_{j=1}^{i} m_i \mathbf{b_i}$

$V := V \cup \{v\}$

end for

Choose the first ℓ v's in V such that the minimal sign changes occur with the length of the vector being the tie-breaker and that the ℓ vectors form a basis and store in $\mathcal{B}^*$

Return $\mathcal{B}^*$

After all the sign patterns are generated, order them in accordance to minimal sign changes. Choose the best ℓ vectors that form a basis for the null space of E and use these vectors in Step 4 of Algorithm 2.

4.3.2. Results of Heuristics. With these two heuristics put into place, one can analyze the results of the timing of the Khovanskii-Rolle continuation algorithm with how well the choices made via the heuristics in the pre-conditioning actually compare.

For the previous example, all possible parameterizations of the monomials and two sets of basis vectors were tested with the Khovanskii-Rolle software [12]. The two sets of basis vectors are

$$N_1 = \{[-1,-1,2,-2,1,2,1],[4,4,2,3,3,1,3]\} \text{ and } N_2 = \{[3,3,4,1,4,3,4],[4,4,2,3,3,1,3]\}.$$

For convenience, let $A = a^{-1}b^2c^2d$, $B = ac$, $C = bc^4d^4$, $D = d$, $E = e$, $F = b^2c$, and $G = b^{-4}c^{-7}d^{-5}e^{-1}$ and write AB for the parameter assignment $s_1 = A, s_2 = B$.

The several runs marked as "fail" failed to run in the software on most or all of the 10 runs in the corresponding cells of Table 4.1.

The overall fastest timing was BG for the choice of parametrization using N_2 as the choice for the annihilating vectors. Our heuristic for choosing a parameterization picked out BC, and we would choose N_2 since all entries of both vectors are positive, i.e., no sign changes. However, this is clearly a heuristic and certainly could result in less-than-optimal choices.

4.3.3. Alternatives for Heuristic Choices. Another option is to consider the curvature of the curves to be traced than to consider the polygon in which they reside. However,

TABLE 4.1. Average run time of 10 runs for each possible parameter assignment and two choices of null vectors. Columns give two choices of vector space basis. Rows give choices of parameterization. Note: The failed runs were due to poor numerics.

Assignment	time for N_1 (sec)	time for N_2 (sec)
AB	37.78	31.52
AC	62.77	60.93
AD	273.94	fail
AE	30.05	31.48
AF	36.24	31.89
AG	57.58	57.34
BC	32.82	32.44
BD	42.3	45.13
BE	fail	fail
BF	46.73	39.17
BG	34.84	29.96
CD	fail	112.16
CE	59.09	59.51
CF	62.26	68.07
CG	56.34	57.46
DE	52.05	56.87
DF	105.09	fail
DG	47.43	57.78
EF	41.94	44.57
EG	47.59	54.87
FG	44.69	49.16

it would likely be significantly more expensive to consider optimizing curvature than to simply proceed with any choice of monomial order.

Furthermore, one could consider scaling the parameterizations to decrease the coefficients of the master functions and Jacobian determinants which are some of the curves traced in Khovanskii-Rolle. Such scaling of parameterizations has been considered to good effect previously in some other contexts using SCLGEN [57].

4.3.4. GENERALIZATION TO $\ell > 2$. This generalization is quite straightforward. It is simple enough to enumerate all vertices and compute all angles between the edges meeting

at any given vertex. The rest of the Khovanskii-Rolle algorithm carries up to the $\ell > 2$ case except for the preconditioning done in Algorithm 4. For that step, simply construct an $(\ell - 1)$-dimensional mesh on the portion of the appropriate hyperplane within the polytope. When doing so, one must then compare the angle between the correct linear subspaces that correspond to each vertex. We note that careful implementation must be done in order to properly identify the angle as opposed to its supplementary one. In addition, the various sanity checks (see [12]) that are employed for the $l = 2$ case for the Khovanskii-Rolle algorithm must be properly considered for the general case. We conclude that the $l > 2$ case is beyond the scope of this dissertation and is the subject of further work.

4.4. Gale Transform without Khovanskii-Rolle

Nothing requires that Khovanskii-Rolle needs to be used to compute the solutions of the original fewnomial system. After applying Algorithm 2 to a fewnomial system, regular homotopy continuation can be applied to solve for all the complex solutions of the gale dual system. These solutions can then be unwrapped via Algorithm 3, and normal numerical methods can be employed for determining which solutions are "real".

4.5. Software: `galeDuality`

`galeDuality` is a software package for studying fewnomial systems by performing the symbolic gale transform. `galeDuality` can perform the gale transform in both directions, i.e., it implements both Algorithm 2 and Algorithm 3. Also, either Khovanskii-Rolle Continuation [12] or standard homotopy methods may be used. In addition, the software package `alphaCertify`[43] is used to numerically certify the solutions so as to properly handle the

gale transformation's numerical errors. A user's manual for `galeDuality` is provided in Appendix B and indicates installation instructions, types of input files, and general use.

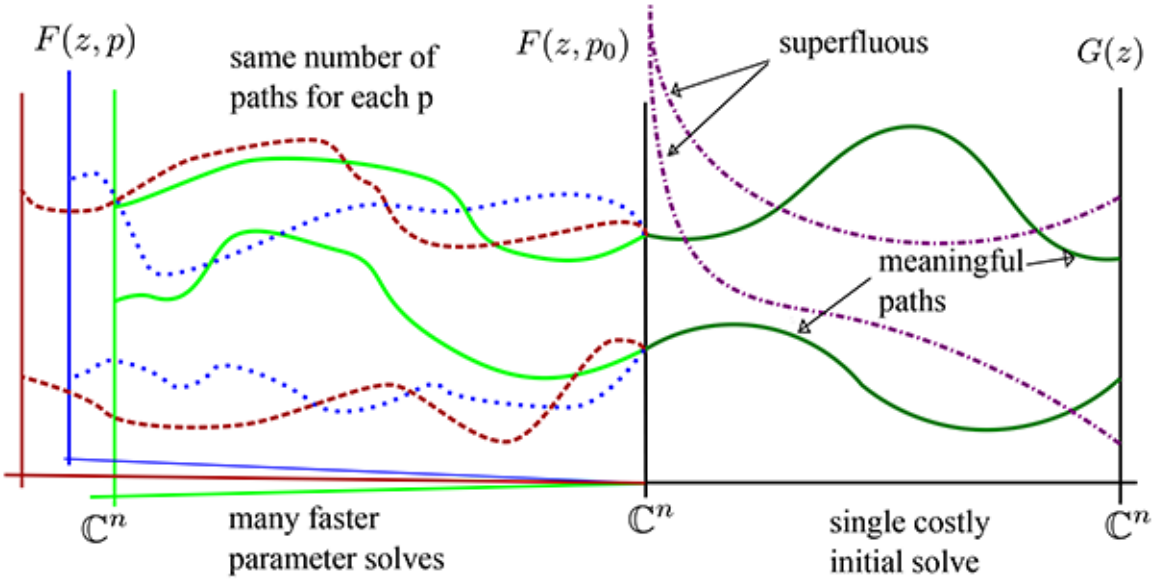
CHAPTER 5
Parameter Homotopies

5.1. Types of Start Sytems

One of the fundamental units for determining complexity in Numerical Algebraic Geometry is counting the number of paths tracked in order to find the solutions of a polynomial system. In general, many polynomial systems are deficient [52], and it is beneficial, to avoid wasted computations, to reduce the number of extraneous paths that track to infinity. In many applications, one wants to solve the same system of polynomials repeatedly but with many different coefficient values, often for many parameter points of interest. Let M be the number of parameter points. One could, in principle, use any type of start system to solve this system at each of these points of interest. Suppose that a polynomial system has N solutions generically and that a particular start system has P paths. If the number of extraneous paths is not reduced, then $(P - N)M$ extra paths are tracked.

From the most number of paths to the least number of paths we have total-degree, linear product, general linear product, polyhedral, and coefficient-parameter start systems. In general, the computational complexity to create a start system is inversely proportional to the number of paths that the start system has. In other words, total-degree start systems are the cheapest and coefficient-parameter start systems are the most expensive to create. It should be noted that both polyhedral methods and coefficient-parameter homotopies give a sharp bound when the coefficients are chosen generically.

In practice, we have found that it is beneficial to create a generic coefficient-parameter start system from a total-degree start system.



However, it is known that with probability 1, the number of paths that a coefficient-parameter start system and the number of solutions of a generic system is the same [61] and [53]. Let $g(z,p)$ be the start system for a total degree homotopy and let $f(z,p^*)$ be the target system with p^* chosen generically. We now construct a coefficient-parameter homotopy from $f(z,p^*)$ to $f(z,p_i)$, for each p_i that is a parameter point of interest.

When one wants to solve the same system with different parameter values, this problem becomes *embarrassingly parallelizable*. That is, computing the solutions of the system at each parameter value of interest is independent of any other parameter value. Each independent problem can then be placed on a separate processor. The following sections describe the general algorithm and the software, `paramotopy`, that implements it.

5.2. `paramotopy`

`paramotopy` is a C++ program designed to solve parameter homotopies in parallel that utilizes the `bertini` [13] library as its computational engine. In this chapter, we provide the

main mathematical algorithm, Algorithm 6, pseudocode for the fundamental algorithm, discuss how path failures are managed automatically, and provide the technical details on both parallelization and data management in `paramotopy`. The parallelization of `paramotopy` uses a standard master/slave paradigm, where a master hands out work for the slaves to do. After a slave has finished a job, the slave reports back to the master and receives new work until all tasks have been exhausted.

It is of import to note that `paramotopy` is not the first homotopy continuation software to implement parameter homotopies. `bertini` has this option and so does `Phcpack`. `paramotopy` is the first such software to efficiently perform parameter homotopies in parallel.

5.3. Main algorithm

We first present the main parameter homotopy algorithm that is implemented in `paramotopy`. Note in particular the input value K and the while loop at the end, both included to help manage path failures during the Stage 2 runs.

To find all solutions for all $p \in L$, we must have that all solutions of $F(z, p_0)$ are nonsingular. This is because we can only follow paths starting from nonsingular solutions during the parameter homotopies following the first run. Deflation could be used to regularize singularities in Stage 1 before beginning Stage 2, but this is not currently implemented. It is possible to have singular solutions at a randomly chosen point $p_0 \in \mathcal{P}$, though this would indicate with probability one that there will be a singular solution for a Zariski open set of points in $\mathcal{P}$. At worst, `paramotopy` will find all solutions at each parameter value $p \in L$ except those coming from the singular points at p_0. Singular solutions for p_i, $i > 0$, i.e., other than at the starting point p_0, are not a problem and will be found, thanks to the use of endgames.

Algorithm 6: Main algorithm.

Input: $F(z;p)$, a set of polynomial equations in N variables $z \in \mathbb{C}^N$ and M parameters $p \in L \subset \mathcal{P} \subset \mathbb{C}^M$; $\ell = \mid L \mid$ parameter values at which the solutions of $F(z;p)$ are desired; bound K on the number of times to try to find solutions for any given $p \in L$, in the case of path failures.

Output: List of solutions of $F(z;p) = 0$ for each $p \in L$

Choose random $p_0 \in P$;

Solve $F(z;p_0) = 0$ with any standard homotopy. (Stage 1);

Store all nonsingular finite solutions in set S;

Set $\mathcal{F} := \phi$. (Beginning of Stage 2.);

for *i=1 to* ℓ **do**

Construct parameter homotopy from $F(z;p_0)$ to $F(z;p_i)$;

Track all $|S|$ paths starting from points in S;

Set $\mathcal{F} := \mathcal{F} \cup \{i\}$ if any path fails;

end

(End of Stage 2.);

Remark: `paramotopy` does not currently check the quality of the Stage 1 results. While theory dictates that some paths converge while others diverge as $t \to 0$, the reality is that paths can fail for numerical reasons. For example, the path tracker can jump between paths if two paths become very near. This is largely mitigated in `bertini` via adaptive precision and a check at $t = 0.1$ (or some other user-defined setting) as to see whether all paths are still distinct. One would have to use path-certification at every every time step, as done in [14], as a way to remove the possibility of path-crossing occurring. This technique is computationally prohibitive, and it works well enough in practice to verify the uniqueness of paths at $t = .1$ and, if desired, certify solutions at the end [43]. Thus, the user should

Algorithm 7: `paramotopy` Implementation of Main Algorithm.

Load input file and user preferences (otherwise use default preferences);

User modifies settings for run;

Write `config` and `input` sections of `bertini` input file (Stage 1);

`system()` call to `bertini`, in parallel if available (Stage 1);

User ensures quality of Stage 1 results (see remark below);

if *parallel* **then**

`paramotopy` calls `system('mpilauncher step2')` Process command string – files to save, and other parameters;

Load preferences;

if *id==0* **then**

`master()` (Algorithm 8);

end

else

`slave()` (Algorithm 9);

end

Return to `paramotopy`;

end

else

Run Stage 2 internally, single process performing both master and slave;

end

`FailedPathAnalysis()` ;

consider reviewing the output of Stage 1 before launching Stage 2. If nothing else, a rerun of Stage 1 at multiple generic parameter points can significantly increase confidence in the results.

Algorithm 8: `paramotopy` master function.

Create input file in memory;

Get `start` points from completed Stage 1;

MPI_Bcast bertini `input` file and `start` file to each worker;

Get first set of points, from mesh or user-provided file;

Distribute first round of points;

while *points left to run* **do**

    MPI_Receive from any source;

    Create next set of parameters;

    MPI_Send new parameters to `slave()` (Algorithm 9);

end

for *each worker* **do**

    MPI_Send kill tag;

end

Write timing data to disk;

Delete temp files;

5.4. Handling path failures during Stage 2

If a path fails during a Stage 2 run for some parameter value $p \in L$, `paramotopy` will automatically attempt to find the solutions at p by tracking from a different randomly chosen parameter value $p' \neq p_0 \in \mathcal{P}$. It will repeat this process K times, with K specified by the user (why a default of $K = 3$). This is the content of the while loop at the end of the Main Algorithm.

The idea behind this is that paths often fail for one of two reasons:

(1) the path seems to be diverging, or

Algorithm 9: `paramotopy` slave function.

```
MPI_Bcast to receive start and input files from master;
Call bertini executable using Start and Input, to seed memory structures;
Collect all .out files created from the bertini executable;
Inform master ready for work;
Receive initial round of work;
Write .out files;
while Have not received kill tag from master do
    while Have points to solve do
        Modify num.out file for use in the straight line program;
        Call bertini library;
        Read selected output files into buffer;
        if bufferthreshold exceeded then
            Write corresponding file buffer to disk;
            Clear buffer;
            Initialize buffer;
        end
    end
    MPI_Send to master that work is completed;
    MPI_Receive from master, work or kill tag;
end
Write timing data to disk;
```

(2) the Jacobian matrix becomes so ill-conditioned that either the steplength drops below the minimum allowed or the precision needed rises above the maximum allowed.

For parameter homotopies, there should be no path failures of the first type, unless the scaling of the problem results in solutions that are large in some norm, e.g., $|z|_\infty > 10^5$ in

the current version of `bertini`. That is easily fixed by rescaling the system or adjusting the threshold `MaxNorm` in `bertini`.

For the second type of path failure, the ill-conditioning is caused by the presence of a singularity $b \in \mathcal{B}$ near (or on) the path between p_0 and p. By choosing new starting point p' "adequately far" from p_0, it should be feasible to avoid the ill-conditioned zone around b unless b is near the target value p. In this last case, it is unlikely that choosing more and more new start points p' will have any value, which is why we have capped the number of new starting points allowed at K.

For now, the new point p' is chosen randomly in the unit cube. Future work will detect where in parameter space the failures have occurred, and bound p' away from this region.

Since it cannot easily be determined which paths from p' to p correspond to the failed paths from p_0 to p, there is no choice but to follow all paths from p' to p. To find all solutions at p', we simply use a parameter homotopy to move the solutions at p_0 to those at p'. Of course, if there are path failures, we must choose a different p' and try again.

To demonstrate the capabilities of `paramotopy` to deal with path failures, presented here are the results of a run using the 'cube' system:

$$x^6 + y^6 + z^6 - 1 = 0 \tag{4}$$

Treating x, y as parameters, and letting z be the sole variable, we get a one-variable system. See Input A.1 in the Appendix for the associated paramotopy input file.

Discretizing $[-1.5,\ 1.5]^2$ in a 200x200 grid, for a total of 40,000 points, and solving using the default `bertini` configuration, exactly four parameter points result in path failures: $(x, y) \in \{(0.9425,\ -0.9726),\ (0.9275,\ -0.8826),\ (-1.1536,\ 0.7615),\ (-1.1236,\ 0.8225)\}$.

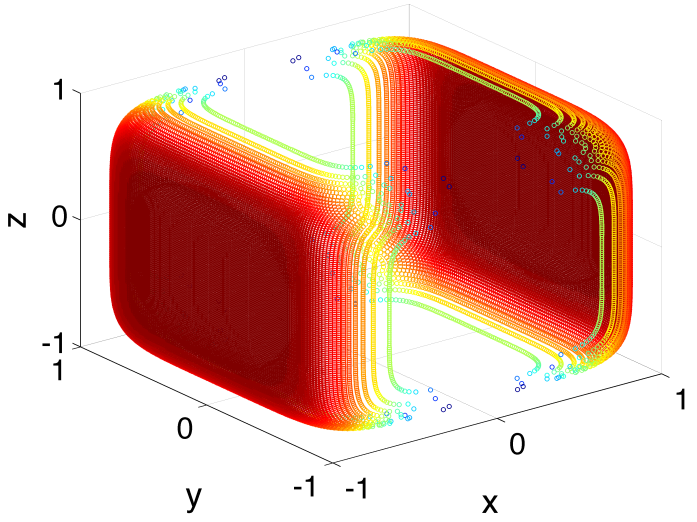


FIGURE 5.1. Plot of samples of $x^6 + y^6 + z^6 - 1 = 0$. This system encounters some path failures using `bertini` default settings, and all these failures can be overcome post-solve using `paramotopy`'s path failure analysis.

`paramotopy` re-solved these points, by first moving the random complex start parameter values to another point p_0, and then tracking to these four points, using a larger `maxnorm` (a bound on the maximum infinity-norm of a solution at any point during a `bertini` solve), and by setting `securitylevel:1`. Of course, whether to move to a new p_0 is determined by the user, as are the tolerance values and other `bertini` settings. A plot of the solutions found, concatenated with the parameter values used to give a triple of coordinates, is presented in Figure 5.1.

One of the benefits of having automatic path failure detection is speed. Solution of the system for all parameter values at the tighter tolerances would have taken much longer; it can be more efficient to run at lower tolerances for a first pass, and re-solve with more secure settings only when and where needed.

5.5. Parallelization and Data Management

One of the features of `paramotopy` that sets it apart from `bertini` is the use of parallel computing for multiple parameter homotopies. `bertini` includes parallel capabilities for a single homotopy run, but not for a series of runs. Parallelization of `paramotopy` was achieved using the master-slave paradigm, implemented with MPI. A single process controls the distribution of parameter points to the workers, which constitute the remainder of the processes. Workers are responsible for writing the necessary files for `bertini`, and for writing their own data to disk.

`bertini` creates structures in memory by parsing an `input` file. As `input` is interpreted, several other files are created, which contain the straight line program, coefficient values, variable names, etc. Since the monomial structure of the polynomials in each Stage 2 run is the same, almost all of these files are identical from one run to the next, so almost all this parsing is unnecessary. The only file that needs to be changed between runs is the file containing parameter values. Parsing requires a significant amount of time especially when compared to the short time needed for parameter homotopy runs, and since we call `bertini` repeatedly, we eliminate as much of this parsing as possible. We do so by calling various functions of `bertini` through a library, preventing extraneous calls that are not needed (e.g. repeated parsing of an `input` file) and by preserving the necessary structure of temporary files.

As `bertini` runs through the paths of a homotopy, it records path-tracking information data files. To prevent an explosion in the number of files needed to contain the data from the `paramotopy` run, the `bertini` output data is read back into memory, and dumped into a collective data file. The collective data files have a maximum size (the default of which

is 64MB), which is user set, and after which a new file is written. On modern systems, in principle all data could be collected into a single file, but transfer of data out of a computing cluster can be cumbersome with extremely large files.

Repeated writing and reading is taxing on hard drives, and clogs a LAN if the workers are using network drives. To free workers from having to physically write temporary files to electronic media storage, an option is provided to the user to exploit a shared memory location (or ramdisk), should it be available. The default location for this is `/dev/shm`. This is commonly available on Linux installations such as CentOS.

As scientific software, efficiency and performance are important. A natural question to answer is what type of speedup is achieved when parallelizing paths. To this end, `paramotopy` is written with the most accurate timing statements possible built in – those of the OpenMP package are used, specifically `omp_get_wtime()`, having extremely fine resolution, depending on the operating system. This enables analysis of performance by process type. If the user is not interested in timing, the program may be compiled without the timing statements by making use of compiler flags.

For the biochemical example considered later in §6.2, we see that we have near linear speedup under the current implementation with only one managing processor. Other levels of parallelization can be done in order to increase efficiency that would incorporate the underlying topology of a computer cluster. This entails using a hybrid openMP and MPI implementation as well as properly using threads. Using threads for homotopy continuation has been considered by Verschelde [79], and it is advantageous to use them. openMP is designed for shared-memory architectures, and MPI is well-suited for distributed memory

architectures. In practice, most clusters are a hybrid of the two, and implementation should attempt to reflect this and needs to be done with care.

5.6. Reduction in the Number of Paths

Besides parallelizing each parameter point, a key feature of using coefficient-parameter homotopies is the reduction in the number of paths. In the biochemical reaction example explored in §6.2, the total degree is 128, but for a generic point in the parameter space, the number of solutions is only 8. If one wishes to solve this particular system at 100,000 parameter points (not an exaggeration), by using parameter homotopies, the number of paths reduces from being $12,800,000$ to only $800,128$ which is slightly more than 6.25% of the number of paths needed to track if the system was solved from scratch for each parameter point. Another system in which the number of paths is reduced is derived from a receding horizon optimization problem, concerned with driving a nonlinear Duffing oscillator to rest [32] and depends on parameters. Here, once more, the system is deficient, in that the number of paths at a generic point is quite a bit less than the total degree (9 versus 132). If we were to solve this system for many different parameter values, we would have a reduction of nearly 93% on the number of paths for each parameter point when we use a coefficient-parameter start system versus a total degree start system.

5.7. Front ends and back ends

Real-world problems may involve many parameters. This could be problematic for the discretization of a parameter space into a uniform sample. Hence, `paramotopy` contains support for both linear uniform meshes of parameters, as well as a user-defined set of parameter

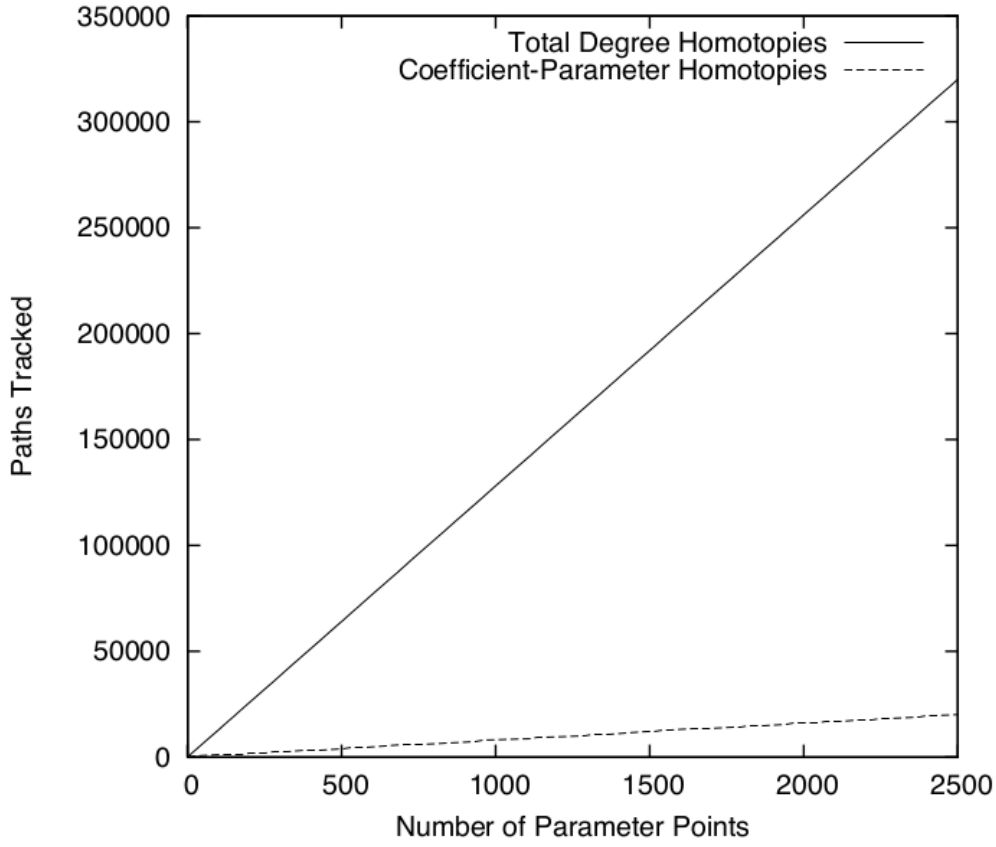


FIGURE 5.2. Reduction in the number of paths by using parameter homotopies.

values stored in a text file. Systems with few parameters can make excellent use of computer-generated discretizations. In contrast, systems with many parameters could use the Monte Carlo sampling method to collect useful information. To use this functionality, the user must generate a text file containing whitespace-separated real and imaginary pairs for each parameter, with distinct parameter points being on separate lines. While computer generated regular meshes are used in the application to rock magnetism for one algorithm,both a user-defined set of points and a mesh are employed in the rock magnetism example that is solved using *decoupling* (§6.3.1).

A generic `MatLab` interface for gathering, saving, and plotting data from an arbitrary `paramotopy` run is provided on the websites of the authors. It can handle both the mesh-style parameter discretizations, as well as user-defined sets. The mesh-style runs generate an n-dimensional array, with the number of solutions, number of real solutions, and solution values stored accordingly. Because the user-defined runs may not emit such a convenient method of storage, a 1D array is created, with line number from the parameter file corresponding to the index in the `MatLab` data structure.

Regarding plotting in this `MatLab` function, two and three dimensional data sets are plotted automatically. Higher dimension data can be plotted with the user's choice of variables appearing on the axes, with a variable used for coloring the points if desired, for display of up to 4 dimensions. Additional basic display techniques included are movie-making and display of the number of real solutions in parameter space. Details on how to use these features may be found in the `paramotopy` user's manual, available from the websites of the authors. Complete documentation of the program and associated classes can be found at http://docs.paramotopy.com.

5.8. Further Development

`paramotopy` is under ongoing development, and the authors of the program expect several extensions in the future. It should be noted that during the Bertini2.0 development in the coming years, many of the parallelism features of parameter homotopies and the object-oriented nature of the program will be incorporated into portions of the `bertini` library. Much work will be done in trying to hold the necessary information in memory, as opposed to writing to the disk and then reading back into memory, which is a major bottleneck when physical drives are concerned.

The developers intend to expand `paramotopy` to automatically search for and map out boundaries between cells of the parameter space within which the parameterized polynomial system has the same number of solutions. More precisely, the discriminant locus of a parameterized polynomial system breaks the parameter space into cells. For all points within the same cell, the parameterized polynomial system has the same number of solutions. In other words, the boundaries give a numerical approximation for values that lie in the discriminant locus. Figure 5.3 clearly shows the various chambers where the number of real solutions differs as one varies the parameter points.

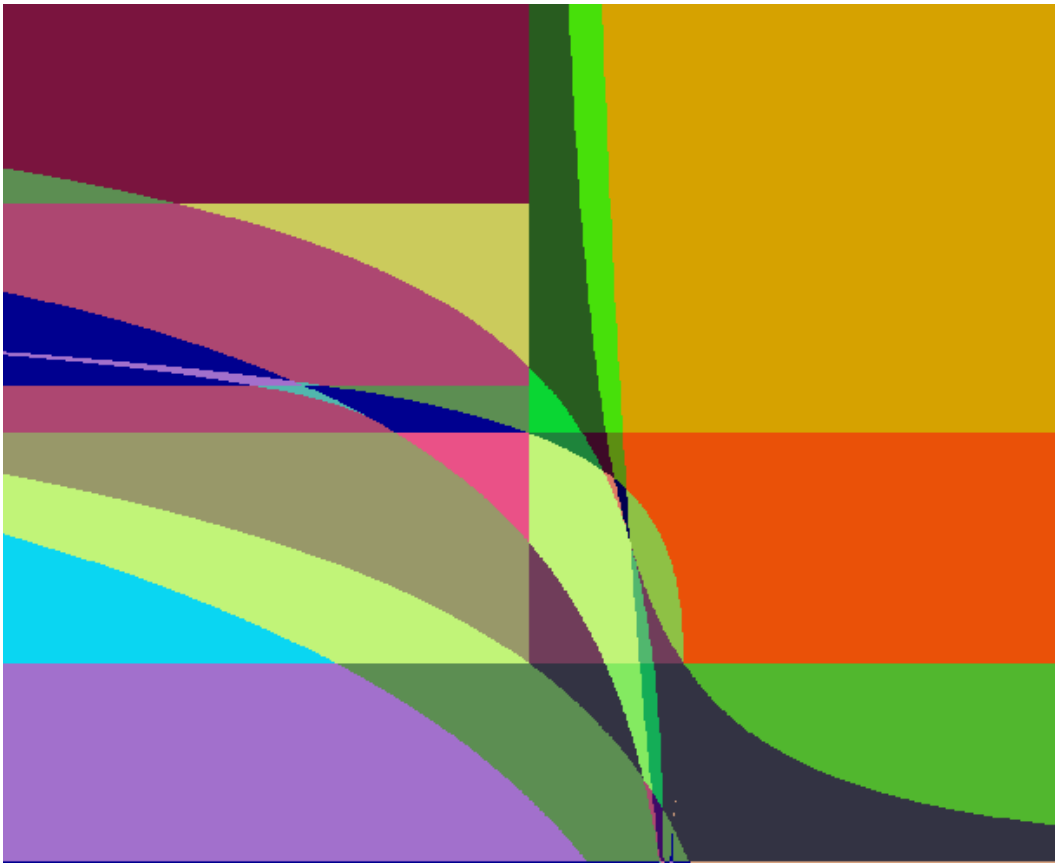


FIGURE 5.3. A polynomial system with 2 parameters, both of which vary from -1 to 1 on a mesh of $1,000$ points apiece. The different colors represent the the number of real solutions in each cell. The boundaries of the chambers are in the discriminant locus.

CHAPTER 6
Applications

6.1. Introduction to Applications

Polynomials appear in many areas of scientific inquiry. In addition, many applications can be viewed as parametrized polynomials. Applications, thus far, have been found in biochemical reactions [25], epidemiology models [62], string theory [55] and [44], statistical mechanics [18], optimal control theory [2, 5, 31], robotics [20], rock magnetism [64], economics, computer vision [76], amongst many others.

In this chapter, we will consider an application in trying to find multiple stationary points in biochemical reaction networks (§6.2) and another situation that arises in rock magnetism (§6.3).

6.2. Multistability in Biochemical Reactions

In certain settings, biochemical reactions can be modeled using mass-kinetic equations [24]. The first order derivatives of these equations are parametrized polynomials. The reaction rates are the parameters in this setup, and they are typically both unknown and expensive to detect via experiments. Thus, numerical simulation is of great value in trying to determine these rates. A natural question that arises is under what conditions does multistability occur for certain choices of parameters. Recent theory provides insight as to when multistability cannot occur [25], [24], and [30], but does not guarantee conditions as to when multistability can occur (only that multistability can exist). As such, `paramotopy` provides a means to search through the parameter space of the reaction rates to find when multistability does indeed occur.

Consider

$$
\begin{aligned}
f_1 &= F_a - D_a a - k_4 ac + k_3 g + k_2 f - k_1 ab \\
f_2 &= F_b - D_b b - k_6 b + k_5 cd + k_2 f - k_1 ab \\
f_3 &= F_c - D_c c + k_8 d - k_7 ce + k_6 b - k_5 cd - k_4 ac + k_3 g \\
f_4 &= F_d - D_d d - k_8 d + k_7 ce + k_6 b - k_5 cd \\
f_5 &= F_e - D_e e + k_8 d - k_7 ce \\
f_6 &= F_f - D_f f - k_2 f + k_1 ab \\
f_7 &= F_g - D_g g + k_4 a - k_3 cg
\end{aligned}
$$

which are derived from mass-kinetic equations.

At a generic point in parameter space, this system yields 7 smooth components of dimension 0 and no other components (i.e. 7 isolated nonsingular solutions). By setting 21 of the parameters as constants and allowing 2 of them (k_2 and k_4) initially to be discretized uniformly on the interval from $[0, 1]$ with 100 values for each parameter (creating $10,000$ parameter points in the mesh), we search for parameter values that have more than one steady-state. In this initial search, we find $7552, 100$, and 2348 parameter points with $1, 2$, and 3 real solutions, respectively. See Figure 6.1. Unfortunately, none of these parameter points yield more than 1 positive real solution (negative solutions are not physical solutions).

We consider the 2348 parameter points that yield 3 real solutions and search the parameter space of the other parameters near one of the points to see how the number of real solutions changes in this particular region. As an arbitrary choice, we choose to allow the

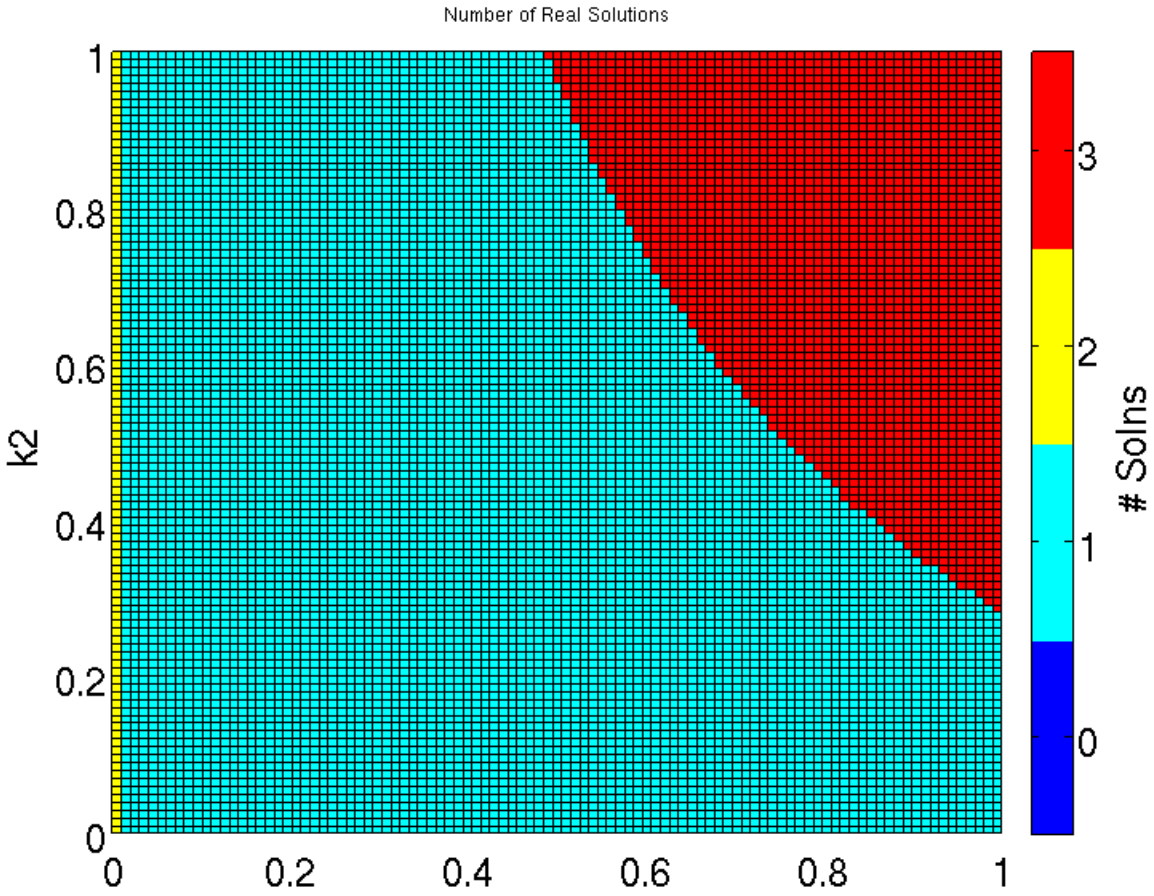


FIGURE 6.1. We see a small region of parameter space where we have 3 real solutions when we vary k_2 and k_4.

k_3 and k_5 parameters vary while letting the k_2 and the k_4 parameters be constant. The reason we do this is that we are looking at the various chambers (with boundaries that is the discriminant locus) where the number of real solutions stays the same. The attempt is to try to find parameter points that are multistationary. Apart from knowing that the number of real solutions changes as we vary parameters, we do not have other methods on how to choose parameter points to increase the number of solutions. As such, we opt to choose parameter values that are in regions of the chambers that are not close to the boundaries.

In other words, we are going to choose parameter values that are not on the discriminant locus but within some cell with the maximal number of real solutions.

After choosing the constants for k_2 and k_4 to hold constant by considering some point in the mesh of parameter values that yielded 3 real solutions, we now vary k_3 and k_5 in the same manner in which we initially varied k_2 and k_4. After this round of computation, we find 4661, 100, 5193, and 46 parameter points with $1, 2, 3$, and 5 real solutions, respectively. See Figure 6.2. Once more, we are not able to find more than 1 positive real solution.

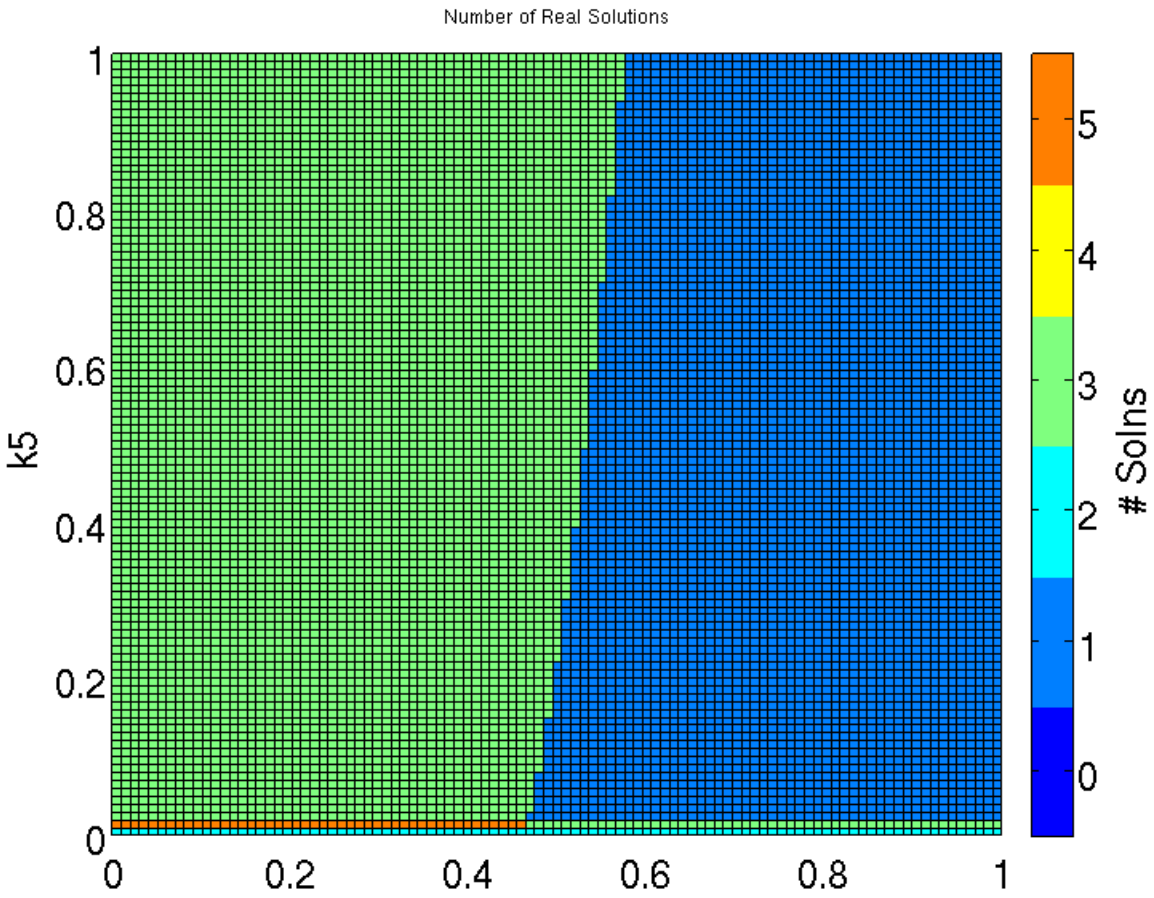


FIGURE 6.2. Holding k_2 and k_4 as constants, and varying k_3 and k_5, we attain parameter points with 5 real solutions.

We now have certain parameter values that yield 5 real solutions, but we want more. We then decide to discretize k_3 by 100 points evenly distributed between $[.005, .015]$ and allow k_5 to vary between $[0, 0.6]$, once more with a uniform mesh of 100 points. This does give us some more parameter values with more than 5 real solutions, but we now only have 100 such parameter points. See Figure 6.3. Alas, no parameter point yields more than one positive equilibria solution.

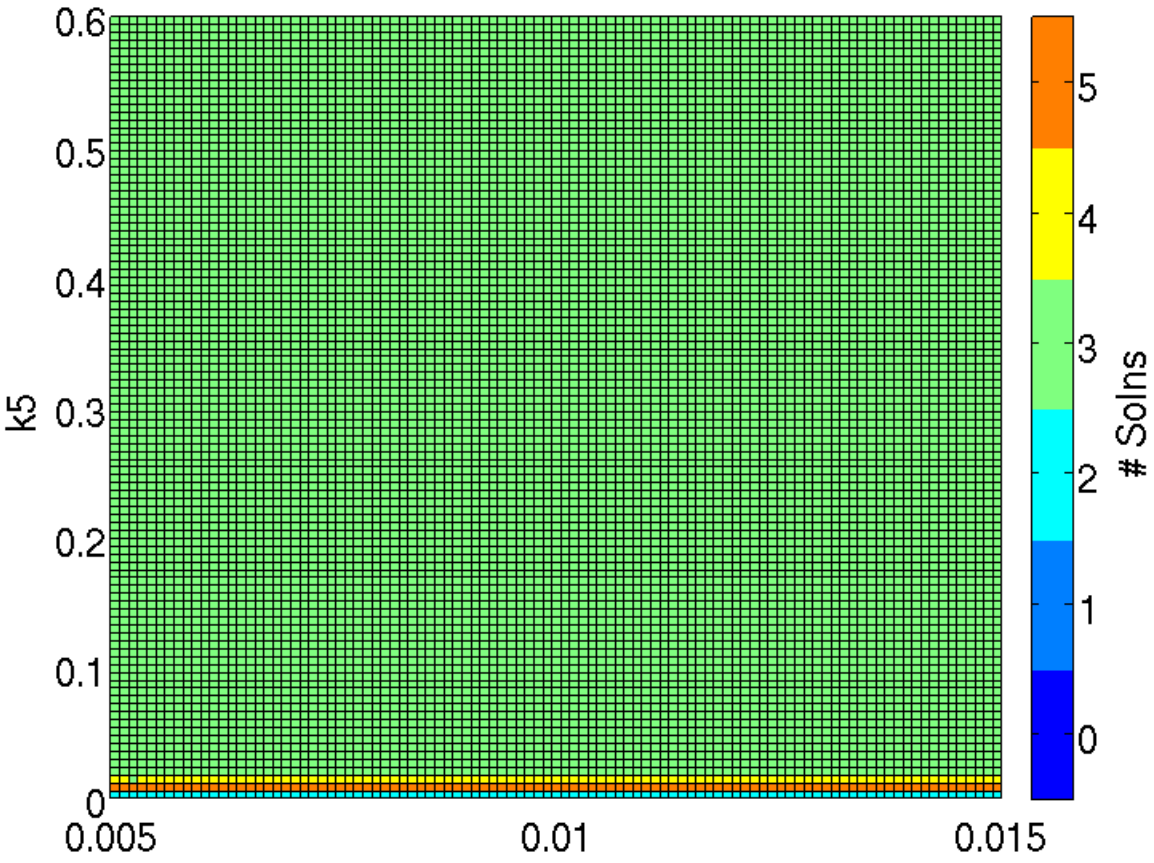


FIGURE 6.3. We vary k_3 and k_5 in a different region of parameter space.

As one last pass, we decide to vary k_3 and k_5 once more, but in a more refined region. We discretize both parameters in $[.003, .03]$ evenly by 100. We find 3000 parameter points

that give us 5 real solutions (see Figure 6.4)! Unfortunately, for all choices of parameters, none of them yield more than 1 biologically realizable solution.

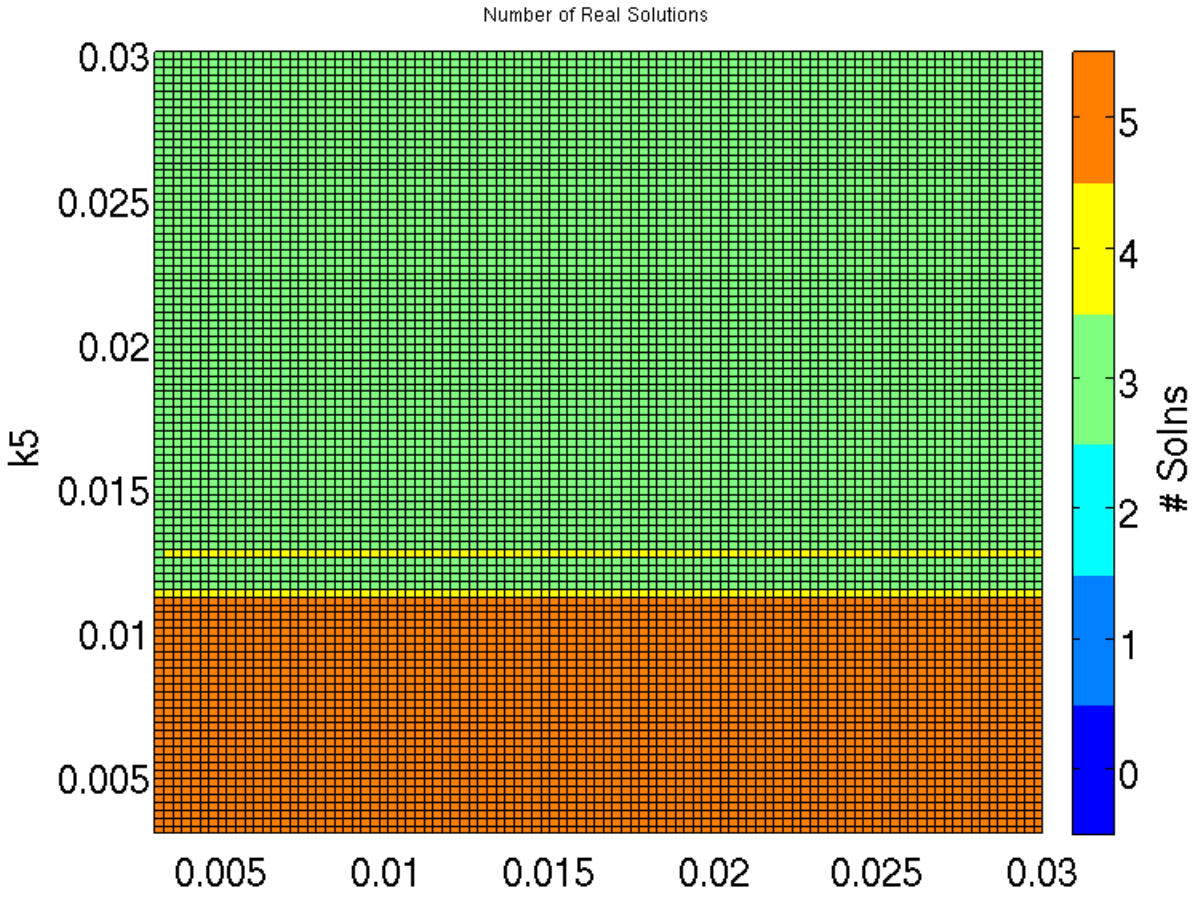


FIGURE 6.4. We choose another region of parameter space to search for multiple stationary points.

By searching through parameter space, we are capable of finding many more parameter points that yield more real solutions by switching which parameters we allow to vary. Unfortunately, our search failed to find parameter values that have more than 1 physically meaning equilibria for this system. This result does not, by any means, show the futility of this type of search as this type of analysis can be applied to other parametrized problems to search for parameter points that have a large number of real solutions.

6.3. Rock Magnetism

Consider a special colony of bacteria that develops an intrinsic magnetic field. These bacteria in the colony form in chains, and one question is how do the magnetic poles switch for each individual bacterium in such a manner so that the magnetic field settles into a stable steady state. Fossils of these bacteria have been found, and the remains can give scientists information about the magnetic field of the Earth at various points in history by treating the Earth's magnetic field as the applied magnetic field to the intrinsic magnetic field produced by the bacteria using models to approximate the overall magnetic field. This problem has been explored extensively in [29] and [64].

Extending the problem, one can consider the bacteria to be individual magnets and no longer in a chain. Let N magnets randomly distributed on the plane with random orientations and different strengths associated with each individual magnet. One can then derive an overarching energy equation that describes the energy of the magnetic field. What stable energy state does the magnetic field move to when an external field is applied to the intrinsic magnetic field? This question is equivalent to computing the hysteresis curve, a difficult problem in magnetism. The area bounded by the hysteresis curve corresponds to the amount of energy necessary to rotate the magnetic domains. As such, the goal in most applications is to minimize this energy loss induced by traversing the hysteresis curve. In the following section, we describe a novel method of computing the hysteresis curve utilizing parameter homotopies. Another method utilizing the structured nature of the polynomial system allows us to attain the solutions orders of magnitude faster than by using conventional techniques, namely the default total degree start system or with regeneration [42].

Various models have been employed to approximate the magnetic field, but we restrict our attention to the Stoner-Wohlfarth model as used in [63] as this model naturally assumes the role of a polynomial system. This model naturally extends to 3 dimensions, but we assume for simplicity all the magnets are on the same plane.

The magnetic energy equation associated with N magnets using the Stoner-Wohlfarth model is

$$E = E_h + E_d + E_a$$

where E_h is the energy of magnetostatic coupling with the external field; E_d is the magnetostatic self-energy (or "demagnetizing energy") of the system; and E_a is the magnetocrystalline anisotropy energy.

The energy of the magnetic moments in an external magnetic field H_0 (the parameter discretized in the algorithm presented in the next subsection), given in Tesla is

$$E_h = -\mu_0 \mathbf{H_0} \cdot \sum_{i=1}^{N} \mu_\mathbf{i}.$$

The energy of interaction between a collection of dipole moments, also called the demagnetizing energy, is equal to

$$E_d = \frac{-\mu_0}{2} \sum_{i=1}^{N} \mu_\mathbf{i} \cdot \mathbf{H'_i}$$

where $\mathbf{H'_i}$ is the magnetic field due to all the dipole moments.

We only consider magnets with uniaxial anisotropy. As such, the magnetocrystalline anisotropy energy is

$$E_a = \sum_{i=1}^{N} V_i(K_1(1 - z_i^2) + K_2(1 - z_i^2)^2)$$

where V_i is the volume of the i^{th} magnet and z_i is the magnetic moment in the z direction, and K_1 and K_2 are parameters for magnetization along the two corresponding axes in the x and y directions of the magnets, respectively.

After translating this energy equation into a polynomial system by taking first order derivatives, we obtain the following generic system of equations where the solutions correspond to the critical points of the magnetic field for N magnets (i.e., $2N$ free-poles) and an applied field

$$
\begin{aligned}
f_{6i+1} &= a(qL(\sum_{j=1,j\neq i}^{2N} x_{x,i,j}x_j + x_{z,i,j}z_j) + x_{x,i,i}x_i + x_{z,i,i}z_i) + x_{H,i,i}H_{x,i} - w_i x_i \\
f_{6i+2} &= a(qL(\sum_{j=1,j\neq i}^{2N} z_{x,i,j}x_j + z_{z,i,j}z_j) + z_{x,i,i}x_i + z_{z,i,i}z_i) + z_{H,i,i}H_{z,i} - w_i z_i \\
f_{6i+3} &= p_{x,i}x_i^2 + p_{z,i}z_i^2 - 1 \\
f_{6i+4} &= a(qL(\sum_{j=1,j\neq i+1}^{2N} x_{x,i+1,j}x_j + x_{z,i+1,j}z_j) + x_{x,i+1,i+1}x_{i+1} + x_{z,i+1,i+1}z_{i+1}) + \\
&\quad x_{H,i+1,i+1}H_{x,i+1} - w_{i+1}x_{i+1} \\
f_{6i+5} &= a(qL(\sum_{j=1,j\neq i+1}^{2N} z_{x,i+1,j}x_j + z_{z,i+1,j}z_j) + z_{x,i+1,i+1}x_{i+1} + z_{z,i+1,i+1}z_{i+1}) + \\
&\quad z_{H,i+1,i+1}H_{z,i+1} - w_{i+1}z_{i+1} \\
f_{6i+6} &= p_{x,i+1}x_{i+1}^2 + p_{z,i+1}z_{i+1}^2 - 1
\end{aligned}
$$

where $x_i, z_i, x_{i+1}, z_{i+1}$ are the magnetic moments of the i^{th} magnet; w_i and w_{i+1} are lagrange multipliers; $a, q, L, x_{x,i,j}, z_{x,i,j}, x_{z,i,j}, z_{z,i,j}, H_{x,i}$, and $H_{z,i}$ are constants. The terms with coefficients of $x_{x,i,j}, z_{x,i,j}, x_{z,i,j}, z_{z,i,j}$, with $i \neq j$, correspond to the interactions between the

magnetic moments i and j, and the terms with coefficients of $x_{x,i,i}, z_{x,i,i}, x_{z,i,i}, z_{z,i,i}$ correspond to the self-interaction of the i^{th} magnetic moment. The third and the final equation impose a condition that the norm of a magnetic moment is 1.

6.3.1. Algorithm for Computing the Hysteresis Curve. We consider two methods for computing the critical points of the magnetic field. The first method is to treat all coefficients as random complex values. Once a start system is solved, parameter homotopies are utilized to solve the system at the specific coefficient values of interest, where these parameter values are determined by location, strength, and the type of anistropy. The applied field parameters $H_{x,i}$ and $H_{z,i}$ are the only coefficients that change from system to system when computing the hysteresis curve. However, we still treat the other coefficients as random, complex parameters to keep the system as general as possible, but for each system these coefficients remain constant throughout. By evenly discretizing the applied field parameters on the interval from -1 to 1, we are capable of finding all the critical points at particular applied field strengths.

The second method is to treat only the applied field parameters as random complex values and set the other coefficients as constants at their appropriate real values initially. In the same manner, we discretize the applied field parameter on the interval from -1 to 1.

6.3.2. Decoupling in Rock Magnetism. In this section, we introduce how we can compute the critical points of the energy equation by utilizing a block system setup. Imagine we have $2N$ magnetic moments and that only two magnetic moments are interacting with one another at a time. In our equations, this corresponds to turning off the interaction terms between the magnetic moments that are not interacting with one another.

As such, one block for two magnetic moments interacting with one another becomes

$$
\begin{aligned}
f_{6i+1} &= a(qL(x_{x,i,i+1}x_{i+1} + x_{z,i,i+1}z_{i+1}) + x_{x,i,i}x_i + x_{z,i,i}z_i) + x_{H,i,i}H_{x,i} - w_i x_i \\
f_{6i+2} &= a(qL(z_{x,i,i+1}x_{i+1} + z_{z,i,i+1}z_{i+1}) + z_{x,i,i}x_i + z_{z,i,i}z_i) + z_{H,i,i}H_{z,i} - w_i z_i \\
f_{6i+3} &= p_{x,i}x_i^2 + p_{z,i}z_i^2 - 1 \\
f_{6i+4} &= a(qL(x_{x,i+1,i}x_{i+1} + x_{z,i+1,i}z_{i+1}) + x_{x,i+1,i+1}x_{i+1} + x_{z,i+1,i+1}z_{i+1}) + \\
&\quad x_{H,i+1,i+1}H_{x,i+1} - w_{i+1}x_{i+1} \\
f_{6i+5} &= a(qL(z_{x,i+1,i}x_{i+1} + z_{z,i+1,i}z_{i+1}) + z_{x,i+1,i+1}x_{i+1} + z_{z,i+1,i+1}z_{i+1}) + \\
&\quad z_{H,i+1,i+1}H_{z,i+1} - w_{i+1}z_{i+1} \\
f_{6i+6} &= p_{x,i+1}x_{i+1}^2 + p_{z,i+1}z_{i+1}^2 - 1
\end{aligned}
$$

We then solve each block independently of one another. A generic start system for the full block system is constructed by combinatorially putting together the solutions from each individual block. We use this start system to move to a target system where the interaction terms between all of the other magnetic moments are turned on all at once.

For instance, the equation f_{6i+1} becomes

$$
\begin{aligned}
&a(qL(x_{x,i,i+1}x_{i+1} + x_{z,i,i+1}z_{i+1}) + x_{x,i,i}x_i + x_{z,i,i}z_i) + x_{H,i,i}H_{x,i} - w_i x_i + \\
&s(a(qL(\sum_{j=1,j\neq i,i+1}^{2N} x_{x,i,j}x_j + x_{z,i,j}z_j)))
\end{aligned}
$$

where s is defined to be $1-t$, since we track from $t=1$ to $t=0$. Similarly, f_{6i+2}, f_{6i+4}, and f_{6i+5} also have the corresponding interaction terms turned on, as well.

In this example, each of the individual blocks has half of the paths diverge to infinity. When combinatorially building the solutions, we attain a start system with the same number of solutions as the system of interest. None of the paths tracked from these starting points diverge to infinity and as such we capture all the solutions. This nicety, in general, is not true. We are not guaranteed that by solving an individual block, having some solutions diverge in that block, combinatorially building a start system from all such blocks built in this manner, and then tracking to the system of interest by turning on interaction terms between the blocks will *guarantee* that we find all the complex solutions for the system. However, due to the highly structured nature of this system, a block system setup seems to work in this case.

It is true that if each block has the total degree number of solutions, then combinatorially building a start system and turning on and off various interaction terms will work. Doing this amounts to nothing more than creating a different start system with the total degree number of paths, but not the total degree start system created by `bertini`, which is a start system that has all starting points that are some root of unity for each coordinate initially somewhere on a circle of a random radius.

6.3.3. Results. The illustrative example of the previous section was run for varying N on a single 2.67 GHz Xeon processor running the CentOS operating system. Table 6.1 provides the average of 10 runs per entry for $N = 2, \ldots, 8$, using a Bézout homotopy, regeneration, the basic decoupling algorithm, and the previously described variant. Timings were stopped at 24 hours.

TABLE 6.1. Average run time of 10 runs for four ways of computing the solutions of the illustrative example – Bézout, regeneration, basic decoupling, and the decoupling variant – for various values of N. Timings in seconds except where stated otherwise.

N	Bézout	Regeneration	Basic Decoupling	Decoupling Variant
2	0.5	0.6	**0.4**	0.2
3	11.0	2.4	**0.8**	0.6
4	268.6	13.2	**1.6**	1.9
5	3757.2	89.0	**3.7**	8.6
6	59131.5	519.9	**18.9**	53.5
7	$>$ 24 hours	3470.7	**105.2**	344.3
8	$>$ 24 hours	18109.6	**903.6**	2269.6

CHAPTER 7
Using Monodromy Action versus Adaptive Multiprecision

At the present time, numerical algebraic geometry software implicitly uses the so called 'γ' trick, which guarantees with probability 1 that the homotopy constructed will not have paths that actually cross. It does not guarantee, however, that the path tracking does not become close to the ill-conditioned areas surrounding these bad points. Current implementations of adaptive multiprecision methods increase precision if the condition number of the Jacobian matrix passes some threshold. The following is an outline of a new algorithm that does not require an increase of numerical precision and as such is not, more than likely, necessarily as computationally intensive.

Consider the following scenario. We have N paths that we are tracking from a start system to a specific target system. Suppose we have a *ramification point* (a parameter value at which there are multiple roots) downstairs, and that m of the N paths track near the corresponding branchpoint upstairs. Instead of increasing precision when the condition number reaches the threshold, a homotopy is constructed to circle around this region, so as to avoid the poor numerics. By circling around the singularity, we know that m paths will permute according to the monodromy action associated with the singularity. We have a bound on the the number of times we must circle the singularity, namely N.

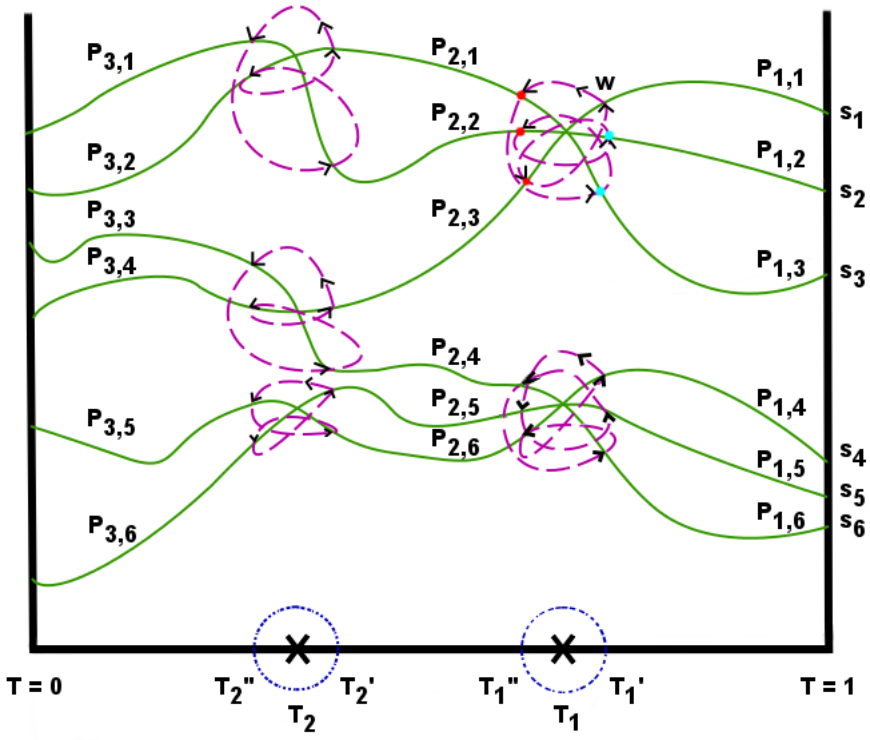


FIGURE 7.1. Schematic representation of the running example. The green curves are the paths to be tracked. They intersect in two triple points in the fiber above $t = T_1$ and in three double points in the fiber above $t = T_2$. The dotted blue circles are the monodromy loops about the two ramification points. The dashed magenta loops around the paths are the fibers above the monodromy loops.

7.1. Proof of Concept

In this section, we describe the method with an illustrative example, represented in Figure 7.1. In this case, there are 6 paths to track from $t = 1$ to $t = 0$, which intersect in two triple points in the fiber above $t = T_1$, then in three double points above $t = T_2$. In fact,

it is easy to write down such a system:

$$\begin{aligned} f_1(x,y,t) &= (x^3-1)\cdot t + (x^3+2)\cdot(1-t) \\ f_2(x,y,t) &= (y^2-1)\cdot t + (y^2+0.5)\cdot(1-t) \end{aligned}$$

Of course, this system was contrived for the purposes of this dissertation. In practice, actual path intersections are exceedingly rare and triple (or more) intersections are even more rare. However, the algorithm proposed in this dissertation applies not only in the situation of actual path intersections but also, much more broadly, in the setting of near path-crossings.

For the moment, we restrict our computational resources to tracking one path segment at a time, on a single processor. The use of multiple processors is described in §7.4.

7.1.1. The first path segment, $P_{1,1}$. Tracking begins with s_1, one of the six start-points at $t = 1$. Tracking forwards (towards $t = 0$), the point w is reached, in the fiber above $t = T_1'$. This is the first point along the path segment $P_{1,1}$ at which the Jacobian matrix is ill-conditioned. This ill-conditioning is recognized as the breaking of an adaptive precision condition [9], at which point a monodromy loop is initiated. This loop is represented in the figure as a blue dotted circle about $t = T_1$. See §7.1.3 for ideas on how to choose the radius of the monodromy loop.

We track around this monodromy loop (following along the magenta dashed curve in the figure) until we arrive back at w. In this case, three trips around the loop are needed. Along the way, we collect all points above T_1'', depicted as red dots in the figure, and those points above T_1' other than w, i.e., the blue dots. At this point, tracking for segment $P_{1,1}$ is complete. In particular, the tracking of segment $P_{1,1}$ consisted of:

- Beginning with s_1 and a direction;
- Tracking until the need for adaptive precision was recognized; and
- Tracking around a monodromy loop until it closed up, collecting various bits of information along the way.

The output of this procedure is a set of five new path segments to be tracked, in addition to the other five initial path segments beginning at $s_2, \ldots, s_6$. In particular, we now have three new forward path segments ($P_{2,1}$, $P_{2,2}$, and $P_{2,3}$) beginning at the red dots, and two new backwards path segments ($P_{1,2}$ and $P_{1,3}$), beginning at the blue dots. Obviously, there is no value in tracking backwards along $P_{1,1}$ from w as we have already tracked along that path.

If a path does not encounter ill-conditioning, we could simply track all the way to $t = 0$. More precisely, we could track to $t = t_{endgame}$, at which point an endgame would take over to complete the path. The algorithm we are proposing is only intended to take the paths successfully from $t = 1$ to the beginning of the endgame, after which the endgames inherently manage ramification points. See [10] and [74] for details on how this is done, and the underlying theory for why such techniques work.

7.1.2. After the first monodromy loop. In our example, we now have five initial forward path segments, three new forward path segments, and two new backwards path segments. Technically, the processor could choose any of these segments to run next. However, choosing either $P_{1,2}$ or $P_{1,3}$ in the forward direction would result in redundant trips around the monodromy loop just handled (or one very close to it, depending on the value of t at which ill-conditioning is first encountered). Thus, the algorithm calls for handling all backwards path segments before handling forward path segments.

In following $P_{1,2}$ backwards from the blue dot, we eventually reach startpoint s_2 above $t = 1$. Once we reach $t = 1$, we can simply search our list of path segments to track forwards and eliminate s_2. After treating $P_{1,3}$ similarly, we are left with only the three red dot forward paths, $P_{1,4}$, $P_{1,5}$, and $P_{1,6}$.

If $P_{2,3}$ is tracked next, we encounter a monodromy loop about T_2, beginning at point T_2'. This will set up $P_{2,4}$ as a backwards path segment, resulting in a "backwards" trip around the monodromy loop about T_1, beginning with $t = T_1''$. It is trivial to adapt the idea of tracking forwards through a monodromy loop to this setting, so we do not provide details here. Note, though, that this backwards tracking will result in backwards tracking along $P_{1,4}$, $P_{1,5}$, and $P_{1,6}$, killing off the three remaining forward paths. While there is no clear computational benefit to following $P_{2,3}$ before following, say, $P_{1,4}$ forward, it is amusing to note that the result of chossing $P_{2,3}$ would be that we would ultimately track forward from only one of our startpoints (s_1), the other five being eliminated by backwards path segments caused by monodromy loops!

Continuing in this way, it should be clear that we will ultimately track in some direction along each path segment $P_{i,j}$ and track around five monodromy loops (three trips around each of two triple intersections, two trips around each of the other three double intersections). This is opposed to trying to track from $t = 1$ to $t = 0$ on the six green paths. A brief analysis of the cost of carrying out this algorithm is provided in §7.3.

7.1.3. Choosing the radius of the monodromy loop. When tracking a path segment, say $P_{1,1}$, it is easy to detect the need for a monodromy loop. However, *a priori*, we do not have any knowledge about the actual location of the nearby ramification point (though the sweeping method of [66] might be of use). For that matter, it might be that sheets come

close but never actually intersect, meaning there is actually no ramification point nearby, though there is ill-conditioning.

There are any number of games that could be played to choose monodromy loops (or other shapes, e.g., diamonds), and a full analysis of these options is beyond the scope of this paper. Such an analysis will be much simpler once there is software for testing out these heuristics, i.e., when the development of Bertini 2.0 is far enough along for a careful implementation.

In the meantime, we propose the following heuristic. Since ill-conditioning has not been encountered along $P_{1,1}$ before reaching w, the step size of the path tracker should be reasonably large. Take that as the radius of the monodromy loop. In particular, if h is the current tracker step size coming in to w, the monodromy loop could be

$$t = (T_1' - h) + he^{i\theta},$$

for $\theta = 0, \ldots, 2\pi$.

Of course, if ill-conditioning was noticed because the path just happened to brush the edge of an ill-conditioning zone off to the side of the path, with a ramification point lying in a direction orthogonal to the direction of the path, then this loop would immediately result in ill-conditioning along the monodromy loop. This situation is discussed in more detail in the following subsection.

7.1.4. Handling illl-conditioning along the monodromy loop. The assumption is that we are able to track around the monodromy loop without encountering ill-conditioning. Of course, this might not be the case, and the Jacobian might become ill-conditioning at any point along the loop. If that occurs, there are a few options, listed in order of lowest quality to highest:

(1) Try to build a new monodromy loop off of the current loop in order to avoid this ill-conditioning. This clearly cascades into a debacle, since we need to track around exactly the same loop for each pass (lest the monodromy groups in the fiber get confused by including different sets of ramification points within the loop for different passes about the loop). This is clearly a dangerous, albeit not impossible, option.
(2) Start over with a different loop, e.g., with a loop of larger or smaller radius. This seems to be a reasonable option, though too many reboots of the monodromy loop will eventually eliminate any value gained by avoiding higher precision.
(3) If all else fails, continue to track straight through the zone of ill-conditioning using adaptive precision as necessary. It might be the case that some ramification points are well separated from other ramification points, in which case the monodromy algorithm should work well. Others might be clustered, making adaptive precision the better option.

It seems reasonable to tighten the adaptive precision conditions (by including extra "safety digits," as in [9]) when using these conditions as a trigger for a monodromy loop. Then, by loosening these conditions to a normal level during the loop, there is a better chance that tracking around the loop will succeed. This maneuver essentially tricks the tracker into starting a monodromy loop a bit earlier than with the normal adaptive precision conditions.

As with the radius (and shape) of the monodromy loop, these options will need to be analyzed more carefully once software is readily available. We expect to implement this algorithm in the near future beyond our proof of concept scripts that can be found at http://www.math.colostate.edu/~bates/preprints/mono_comp_page.html.

7.1.5. The necessity of closing the loop. A similar approach was proposed in 1991 [48], before the algebraic geometry underlying polynomial homotopy continuation was well established. In that approach, the authors suggested walking halfway around monodromy loops (called a two-phase homotopy) to avoid ill-conditioning.

While this approach sounds similar in principle, not closing the loop in the fiber is dangerous. In particular, suppose two paths intersect and that, by the vagaries of adaptive steplength, a monodromy loop is triggered on only one of the paths. Due to the monodromy action about the ramification point, the remainder of the path for which a monodromy loop was not triggered will be followed twice, with the remainder of the other path neglected. By closing the monodromy loop as in our algorithm, we never neglect a path and are certain to follow all segments of all paths.

It is worth noting that a monodromy loop can be triggered even if there is no ramification point, i.e., if the solution sheets come close but do not intersect. The result of this false positive will be a seemingly useless monodromy loop for each of the two paths exhibiting ill-conditioning, each traversed just once. While there is theoretically no problem with this scenario, it may seem a waste of computation time. However, the alternative is to increase precision and push through the ill-conditioned zone, which also results in an increase in computation time. Thus, in this setting, seemingly frivolous monodromy loops are not necessarily more costly than the alternative.

7.2. Formal Monodromy Heuristic

Let $f : \mathbb{C}^N \to \mathbb{C}^N$ be a polynomial system, $g : \mathbb{C}^N \to \mathbb{C}^N$ a start system for $f(z)$, and $H(z,t)$ the standard homotopy of §2.2 from $g(z)$ to $f(z)$. A *path segment* is a triple $(p, t_{curr}, t_{target})$ consisting of a point $p \in \mathbb{C}^N$ on a homotopy path with $t = t_{curr}$ and a target value of t, either $t_{endgame}$ or 1, providing a direction. The value of $t_{endgame}$ is the starting point for the endgame [10]. A reasonable default for this (as implemented in Bertini) is 0.1.

Our method consists of a main method (`Main`) and two subroutines (`Track` for regular path tracking and `MonoTrack` for tracking around a monodromy loop).

Algorithm 10: `Monodromy Main`

Input: Homotopy $H(z,t)$ from $g(z)$ to $f(z)$; startpoints s_i for $i = 1, \ldots, k$.

Output: Set E of solutions of $H(z, t_{endgame}) = 0$.

1: Let $P := \{(s_i, 1, t_{endgame}) | 1 \leq i \leq k\}$.

2: **while** $\#P > 0$ **do**

3: Choose a path segment $(p, t_{curr}, t_{target})$ from P, with $t_{target} = 1$, if possible.

4: `Track`$(p, t_{curr}, t_{target})$.

5: **end while**

Algorithm 11: `Track`

Input: Path segment $(p, t_{curr}, t_{target})$.

Output: No output, but list P of path segments to track is updated.

Track from p in the direction of t_{target}, using predictor-corrector methods with adaptive steplength, stopping when either $t = t_{target}$ or an adaptive precision condition has been broken.

Let w be the current point, t_{curr} the current value of t, and h be the latest step size.

if $t = t_{target}$ and $t_{target} = t_{endgame}$ **then**

Add w to set E.

else if $t = t_{target}$ and $t_{target} = 1$ **then**

Remove path segment $(w, 1, t_{endgame})$ from P.

else if an adaptive precision condition has been broken **then**

$its := 0$

while $its < its_{max}$ **do**

`MonoTrack`(w,t_{curr},t_{target},h,its).

if MonoTrack did not report an error **then**

Return.

else

$its := its + 1$.

end if

end while

if $its = its_{max}$ **then**

Continue through the ill-conditioned zone with adaptive precision.

end if

else

Report an error.

end if

Remark on `Track`: The full tracking algorithm is very complicated, so we have opted to refer the reader to the flowcharts of [9] rather than record all of the details here.

Algorithm 12: `MonoTrack`

Input: Point w, t value, t_{curr}, t_{target} for this path segment, and latest step size h, its, list P of path segments (passed by reference)

Output: No output, but list P of path segments to track is updated.

Choose a loop, parametrized by $\theta \in [0, 2\pi]$, using t_{curr}, t_{target}, h, and its..

Let $y := w$.

Let $m := 0$

while $y \neq w$ or $m = 0$ **do**

Track from $\theta = 0$ to $\theta = \pi$, breaking if ill-conditioned is encountered.

Store $(y, t_{curr}, t_{target})$ in P.

Track from $\theta = \pi$ to $\theta = 2\pi$, breaking if ill-conditioning is encountered.

If $y \neq w$, then store $(y, 1 + t_{endgame} - t_{target})$ in P.

$m := m + 1$.

end while

If, at any point, ill-conditioning is encountered, return an error.

Remarks on `MonoTrack`: Rather than tracking around a circle via some parameterization, the standard implementation choice is to inscribe a regular n-gon ($n = 8$ is a common choice) within the circle and follow along the n line segments ($\frac{n}{2}$ for each trip halfway around the circle). Obviously, for our purposes, n must be even.

In line 8, the final argument might look overly complicated. The idea is to have this path segment aimed in the direction opposite of the current direction (towards t_{target}). Indeed,

if $t_{target} = t_{endgame}$, then this final argument becomes 1; if $t_{target} = 1$, then this becomes $t_{endgame}$.

As stated, this algorithm will not pair startpoints with endpoints. In some contexts, that is important, while in others it is not. It may be feasible to add memory to path segments to allow for such a pairing, but this is beyond the scope of this dissertation.

7.3. Cost Analysis

The best case is that ill-conditioning is never encountered during `MonoTrack`. In this case, we replace each ill-conditioned zone needing adaptive precision with a low precision trip around a monodromy loop. While the loop is clearly longer (the circumference of a circle, compared to the diameter or less), the savings from using lower precision should more than compensate for this additional length. Even more compelling is the fact that the monodromy algorithm will avoid path failures due to excessive precision.[1]

The worst case is that ill-conditioning would be encountered during every call to `MonoTrack`, in which case we would resort to adaptive precision tracking on every path *and* incur the additional cost of starting each monodromy loop. This is, of course, far from ideal and serves to increase the computational time for the run with no benefit.

The average case is of course much more complicated. At a minimum, the number of path segments will equal the number of startpoints for the homotopy. The choice of start system dictates this number, though one choice is the total degree start system, for which the number of startpoints is the product of the degrees of the polynomials. The number of path segments encountering ill-conditioning is highly problem-dependent and also dependent on

[1]Bertini sets a maximum level of precision, lest the memory and disk space needs grow too large. Thus, paths can fail with adaptive precision not only at true path intersections, but also in very small neighborhoods about them.

the choice of random numbers in constructing the homotopy. From experience, one might expect between 1% and 10% of path segments will encounter ill-conditioning, though no careful study of this has been conducted.

7.4. Parallelization

Using a single processor, it is easy to see that there will be no path segments accidentally tracked multiple times, in different directions. However, with multiple processors, this could happen.

For example, suppose we attack the example above with six processors. A natural approach would be to give one of the six startpoints to each processor, so that each $P_{1,i}$ is tracked forward. Unfortunately, each of these path segments will trigger a monodromy loop around $t = T_1$, resulting in a great deal of redundancy!

While it is appealing to consider checking whether some given monodromy loop has been considered already, there are two inherent difficulties. First, as is evident in the example above, there can be separate monodromy actions over the same ramification point (or ramification points in very close proximity). While this is not likely at all, it is a concern. Even worse, just because two paths could trigger monodromy loops about the same ramification point does *not* mean that they will necessarily trigger these loops from the same value of t, again due to the vagaries of adaptive step size path tracking. It is useless trying to compare points in the fiber above T_1' when T_1' is path-dependent.

There are two clear approaches for parallelizing this method:

(1) If there are many paths and not too many processors, it is reasonable to accept a bit of redundancy. Since paths (and therefore path segments) are typically sent to processors in batches, not one at a time, one implementation choice is whether to

send a message each time a startpoint is eliminated by backwards tracking. Depending on the parameters of the system and the processors, this may or may not be worthwhile.

(2) Rather than starting all processors on initial path segments $P_{1,i}$, hold some (perhaps most) in reserve, to be used once a monodromy loop has been traversed. This would require significant hierarchical control, but it is not infeasible.

7.5. Future work

This monodromy-based method for avoiding higher precision is still very much at the proof-of-concept level. As we re-develop `bertini` and incorporate this technique, we will analyze the value of the method as a whole and also consider the variety of options for the many subroutines. Also, this method may or may not be useful for specialized types of homotopies. For example, regeneration [42] and parameter homotopies [4] are both useful techniques relying on homotopy continuation for which monodromy could possibly be useful, though this was not considered in this dissertation.

CHAPTER 8
Conclusion

Numerical Algebraic Geometry is an emerging discipline within the mathematical sciences. This dissertation provides an overview of the field in §2.1; introduces the major topics of the dissertation in §3; implements two previously known algorithms in two new software packages `galeDuality` and `paramotopy`, in §4 and §5.2, respectively; and describes two novel algorithms: decoupling (§6.3.2) and monodromy (§7). We also provide applications in biochemical reaction networks §6.2 and rock magnetism §6.3 where the software `paramotopy` can be used as a tool for scientific inquiry. This shows that the software developed is not limited to mathematicians but can be employed by a wide range of scientists in other fields.

In addition, we have also considered two ways to precondition the gale dual that results from the gale transform for the use in the Khovanskii-Rolle Continuation algorithm: 1) choosing an appropriate parametrization of the monomials to make the polytope as "regular" as possible and 2) choosing an appropriate basis for the null space that contains the best sign pattern (§5).

The future of numerical algebraic geometry requires advancing algorithms and increasing the capabilities of the software used in the field. The author of this dissertation will continue to be an active member of the field for years to come.

Bibliography

[1] Eugene L Allgower, Derrick J Bates, Andrew J Sommese, and Charles W Wampler. Solution of polynomial systems derived from differential equations. *Computing*, 76(1-2):1–10, 2006.

[2] Daniel J. Bates, A.G. Beccuti, Ioannis A. Fotiou, and Manfred Morari. An optimal control application in power electronics using numerical algebraic geometry. In *American Control Conference, 2008*, pages 2221–2226. IEEE, 2008.

[3] Daniel J. Bates, Frédéric Bihan, and Frank Sottile. Bounds on the number of real solutions to polynomial equations. *IMRN: International Mathematics Research Notices*, 2007.

[4] Daniel J. Bates, Daniel A. Brake, and Matthew E. Niemerg. Paramotopy: Parameter homotopies in parallel. *submitted to Transactions of Mathematical Software*, 2013.

[5] Daniel J. Bates, Ioannis A. Fotiou, and Philipp Rostalski. A numerical algebraic geometry approach to nonlinear constrained optimal control. In *Decision and Control, 2007 46th IEEE Conference on*, pages 6256–6261. IEEE, 2007.

[6] Daniel J. Bates, Jonathan D. Hauenstein, Matthew E. Niemerg, and Frank Sottile. Computational aspects of gale duality. *in preparation*, 2013.

[7] Daniel J. Bates, Jonathan D. Hauenstein, Chris Peterson, and Andrew J. Sommese. A numerical local dimension test for points on the solution set of a system of polynomial equations. *SIAM J. Numer. Anal.*, 47(5):3608–3623, 2009.

[8] Daniel J. Bates, Jonathan D. Hauenstein, and Andrew J. Sommese. A parallel endgame. *Randomization, Relaxation, and Complexity in Polynomial Equation Solving, eds. L. Gurvits, P. Pébay, J. Rojas, and D. Thompson*, 556:25–35, 2011.

[9] Daniel J. Bates, Jonathan D. Hauenstein, Andrew J. Sommese, and Charles W. Wampler. Adaptive multiprecision path tracking. *SIAM Journal on Numerical Analysis*, 46(2):722–746, 2008.

[10] Daniel J. Bates, Jonathan D. Hauenstein, Andrew J. Sommese, and Charles W. Wampler. *Numerical solution of polynomial systems using the software package Bertini*. SIAM, Philadelphia, PA, 2013.

[11] Daniel J. Bates and Matthew Niemerg. Using monodromy to avoid high precision in homotopy continuation. *Submitted to Theoretical Computer Science*, 2014.

[12] Daniel J. Bates and Frank Sottile. Khovanskii–rolle continuation for real solutions. *Foundations of Computational Mathematics*, 11(5):563–587, 2011.

[13] D.J. Bates, J.D. Hauenstein, A.J. Sommese, and C.W. Wampler. Bertini: Software for numerical algebraic geometry. Software available at `http://www.bertini.nd.edu`, 2006.

[14] Carlos Beltrán and Anton Leykin. Certified numerical homotopy tracking. *Experimental Mathematics*, 21(1):69–83, 2012.

[15] Gian Mario Besana, Sandra Di Rocco, Jonathan D. Hauenstein, Andrew J. Sommese, and Charles W. Wampler. Cell decomposition of almost smooth real algebraic surfaces. *Numerical Algorithms*, 63(4):645–678, 2013.

[16] Frédéric Bihan and Frank Sottile. Gale duality for complete intersections. In *Annales de l'institut Fourier*, pages 877–892, 2008.

[17] Frédéric Bihan, Frank Sottile, et al. New fewnomial upper bounds from gale dual polynomial systems. *Moscow mathematical journal*, 7(3):387–407, 2007.

[18] Kurt Binder and A. Peter Young. Spin glasses: Experimental facts, theoretical concepts, and open questions. *Reviews of Modern physics*, 58(4):801, 1986.

[19] Rafael Bombelli. *L'Algebra*. Italy, 1572.

[20] Daniel A. Brake, Daniel J. Bates, Vakhtang Putkaradze, and Anthony A. Maciejewski. Illustration of numerical algebraic methods for workspace estimation of cooperating robots after joint failure. In *IASTED Technology Conferences*, Pittsburg, PN USA, November 2010.

[21] John Canny and J Maurice Rojas. An optimal condition for determining the exact number of roots of a polynomial system. In *Proceedings of the 1991 international symposium on Symbolic and algebraic computation*, pages 96–102. ACM, 1991.

[22] Gerolamo Cardano. *Book number one about the Great Art, or The Rules of Algebra*. Italy, 1545.

[23] Shui Nee Chow, John Mallet-Paret, and James A Yorke. Finding zeroes of maps: homotopy methods that are constructive with probability one. *Mathematics of Computation*, 32(143):887–899, 1978.

[24] Gheorghe Craciun, J. William Helton, and Ruth J. Williams. Homotopy methods for counting reaction network equilibria. *Mathematical biosciences*, 216(2):140–149, 2008.

[25] Gheorghe Craciun, Yangzhong Tang, and Martin Feinberg. Understanding bistability in complex enzyme-driven reaction networks. *Proceedings of the National Academy of Sciences*, 103(23):8697–8702, 2006.

[26] René Descartes. *La Géomtrié*. Italy, 1637.

[27] Ferenc Domes and Arnold Neumaier. A scaling algorithm for polynomial constraint satisfaction problems. *Journal of Global Optimization*, 42(3):327–345, 2008.

[28] F. J. Drexler. A homotopy method for the calculation of all zeros of zero-dimensional polynomial ideals, 1978.

[29] David J. Dunlop. Magnetism in rocks. *Journal of geophysical research*, 100(B2):2161–2174, 1995.

[30] Elisenda Feliu and Carsten Wiuf. Preclusion of switch behavior in networks with mass-action kinetics. *Applied Mathematics and Computation*, 219(4):1449–1467, 2012.

[31] Ioannis A. Fotiou, Philipp Rostalski, Pablo A. Parrilo, and Manfred Morari. Parametric optimization and optimal control using algebraic geometry methods. *International Journal of Control*, 79(11):1340–1358, 2006.

[32] Ioannis A. Fotiou, Philipp Rostalski, Bernd Sturmfels, and Manfred Morari. An algebraic geometry approach to nonlinear parametric optimization in control. In *American Control Conference, 2006*, pages 6–pp. IEEE, 2006.

[33] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.6.5*, 2013.

[34] C. B. Garcia and T. Y. Li. On the number of solutions to polynomial systems of equations. *SIAM Journal on Numerical Analysis*, 17(4):540–546, 1980.

[35] C. B. Garcia and Willard I. Zangwill. Finding all solutions to polynomial systems and other systems of equations. *Mathematical Programming*, 16(1):159–176, 1979.

[36] Albert Girard. *L'invention nouvelle en l'Algbre*. France, 1629.

[37] Takayuki Gunji, Sunyoung Kim, Masakazu Kojima, Akiko Takeda, Katsuki Fujisawa, and Tomohiko Mizutani. Phom–a polyhedral homotopy continuation method for polynomial systems. *Computing*, 73(1):57–77, 2004.

[38] Wenrui Hao, Jonathan D. Hauenstein, Bei Hu, Timothy McCoy, and Andrew J. Sommese. Computing steady-state solutions for a free boundary problem modeling tumor growth by stokes equation. *Journal of Computational and Applied Mathematics*, 2012.

[39] Wenrui Hao, Bei Hu, and Andrew J. Sommese. Cell cycle control and bifurcation for a free boundary problem modeling tissue growth. *Journal of Scientific Computing*, pages 1–16, 2012.

[40] Wenrui Hao and Shaohong Zhu. Domain decomposition schemes with high-order accuracy and unconditional stability. *Applied Mathematics and Computation*, 2012.

[41] Jonathan D. Hauenstein and Zachary A. Griffin. Real solutions to systems of polynomial equations and parameter continuation. *Advances in Geometry*, 2012.

[42] Jonathan D. Hauenstein, Andrew Sommese, and Charles Wampler. Regeneration homotopies for solving systems of polynomials. *Mathematics of Computation*, 80(273):345–377, 2011.

[43] Jonathan D. Hauenstein and Frank Sottile. Algorithm 921: alphacertified: Certifying solutions to polynomial systems. *ACM Transactions on Mathematical Software (TOMS)*, 38(4):28, 2012.

[44] Yang-Hui He, Dhagash Mehta, Matthew E. Niemerg, Markus Rummel, and Alexandru Valeanu. Exploring the potential energy landscape over a large parameter-space. *Journal of High Energy Physics*, 2013.

[45] Sir Thomas Little Heath. *A History of Greek mathematics*, volume 1. The Clarendon Press, Oxford, 1921.

[46] Birkett Huber and Bernd Sturmfels. A polyhedral method for solving sparse polynomial systems. *Mathematics of computation*, 64(212):1541–1555, 1995.

[47] Birkett Huber and Jan Verschelde. Polyhedral end games for polynomial continuation. *Numerical Algorithms*, 18(1):91–108, 1998.

[48] R.E. Kalaba and L. Tesfatsion. Solving nonlinear equations by adaptive homotopy continuation. *Applied Mathematics and Computation*, 41(2):99–115, 1991.

[49] Askold G. Khovanskii. *Fewnomials*, volume 88. American Mathematical Soc., 1991.

[50] Tsung-Lin Lee, Tien-Yien Li, and Chih-Hsiung Tsai. Hom4ps-2.0: a software package for solving polynomial systems by the polyhedral homotopy continuation method. *Computing*, 83(2-3):109–133, 2008.

[51] Anton Leykin. Numerical primary decomposition. In *Proceedings of the twenty-first international symposium on Symbolic and algebraic computation*, pages 165–172. ACM, 2008.

[52] T. Y. Li, Tim Sauer, and James A. Yorke. The random product homotopy and deficient polynomial systems. *Numerische Mathematik*, 51(5):481–500, 1987.

[53] T.Y. Li, Tim Sauer, and J.A. Yorke. The cheater's homotopy: an efficient procedure for solving systems of polynomial equations. *SIAM Journal on Numerical Analysis*, 26(5):1241–1251, 1989.

[54] Ye Lu, Daniel J. Bates, Andrew J. Sommese, and Charles W Wampler. Finding all real points of a complex curve. *Contemporary Mathematics*, 448:183, 2007.

[55] Danny Martinez-Pedrera, Dhagash Mehta, Markus Rummel, and Alexander Westphal. Finding all flux vacua in an explicit example. *arXiv preprint arXiv:1212.4530*, 2012.

[56] Keith Meintjes and Alexander P. Morgan. A methodology for solving chemical equilibrium systems. *Applied Mathematics and Computation*, 22(4):333–361, 1987.

[57] Alexander Morgan. *Solving Polynominal Systems Using Continuation for Engineering and Scientific Problems*, volume 57. SIAM, 2009.

[58] Alexander Morgan and Andrew Sommese. A homotopy for solving general polynomial systems that respects m-homogeneous structures. *Applied Mathematics and Computation*, 24(2):101–113, 1987.

[59] Alexander P. Morgan. A homotopy for solving polynomial systems. *Applied Mathematics and Computation*, 18(1):87–92, 1986.

[60] Alexander P. Morgan. A transformation to avoid solutions at infinity for polynomial systems. *Applied Mathematics and Computation*, 18(1):77–86, 1986.

[61] Alexander P. Morgan and Andrew J. Sommese. Coefficient-parameter polynomial continuation. *Applied Mathematics and Computation*, 29(2):123–160, 1989.

[62] Philip Munz, Ioan Hudea, Joe Imad, and Robert J. Smith. When zombies attack!: mathematical modelling of an outbreak of zombie infection. *Infectious Disease Modelling Research Progress*, 4:133–150, 2009.

[63] Andrew J. Newell. A high-precision model of first-order reversal curve (forc) functions for single-domain ferromagnets with uniaxial anisotropy. *Geochemistry, Geophysics, Geosystems*, 6(5), 2005.

[64] Andrew J. Newell. Transition to superparamagnetism in chains of magnetosome crystals. *Geochemistry Geophysics Geosystems*, 10(11):Q11Z08, 2009.

[65] John J. O'Connor and Edmund F. Robertson. Mactutor history of mathematics archive. http://www-history.mcs.st-andrews.ac.uk/index.html, February 2014.

[66] K. Piret and J. Verschelde. Sweeping algebraic curves for singular solutions. *Journal of Computational and Applied Mathematics*, 234(4), 2010.

[67] Greg Reid, Ping Lin, and Allan D Wittkopf. Differential elimination–completion algorithms for dae and pdae. *Studies in Applied Mathematics*, 106(1):1–45, 2001.

[68] Greg Reid, Chris Smith, and Jan Verschelde. Geometric completion of differential systems using numeric-symbolic continuation. *ACM SIGSAM Bulletin*, 36(2):1–17, 2002.

[69] Philipp Rostalski, Ioannis A. Fotiou, Daniel J. Bates, A. Giovanni Beccuti, and Manfred Morari. Numerical algebraic geometry for optimal control applications. *SIAM Journal on Optimization*, 21(2):417–437, 2011.

[70] Andrew J. Sommese, Jan Verschelde, and Charles W. Wampler. Numerical decomposition of the solution sets of polynomial systems into irreducible components. *SIAM Journal on Numerical Analysis*, 38(6):2022–2046, 2001.

[71] Andrew J. Sommese, Jan Verschelde, and Charles W. Wampler. Numerical irreducible decomposition using projections from points on the components. *Contemporary Mathematics*, 286:37–52, 2001.

[72] Andrew J. Sommese, Jan Verschelde, and Charles W. Wampler. Using monodromy to decompose solution sets of polynomial systems into irreducible components. In *Applications of Algebraic Geometry to Coding Theory, Physics and Computation*, pages 297–315. Springer, 2001.

[73] Andrew J. Sommese, Jan Verschelde, and Charles W. Wampler. Numerical irreducible decomposition using phcpack. In *Algebra, Geometry and Software Systems*, pages 109–129. Springer, 2003.

[74] Andtrew J. Sommese and Wampler C.W. *The Numerical Solution to Systems of Polynomials Arising in Engineering and Science*. World Scientific, 2005.

[75] Hai-Jun Su, J. Michael McCarthy, Masha Sosonkina, and Layne T. Watson. Algorithm 857: Polsys_glp: A parallel general linear product homotopy code for solving polynomial systems of equations. *ACM Transactions on Mathematical Software (TOMS)*, 32(4):561–579, 2006.

[76] Gabriel Taubin, Fernando Cukierman, Steve Sullivan, Jean Ponce, and David J. Kriegman. Parameterized families of polynomials for bounded algebraic curve and surface fitting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(3):287–303, 1994.

[77] Jan Verschelde. Algorithm 795: Phcpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software (TOMS)*, 25(2):251–276, 1999.

[78] Jan Verschelde, Pierre Verlinden, and Ronald Cools. Homotopies exploiting newton polytopes for solving sparse polynomial systems. *SIAM Journal on Numerical Analysis*, 31(3):915–930, 1994.

[79] Jan Verschelde and Genady Yoffe. Polynomial homotopies on multicore workstations. In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, pages 131–140. ACM, 2010.

[80] Jan Verschelde and Yan Zhuang. Parallel implementation of the polyhedral homotopy method. In *Parallel Processing Workshops, 2006. ICPP 2006 Workshops. 2006 International Conference on*, pages 8–pp. IEEE, 2006.

[81] Charles Wampler. An efficient start system for multi-homogeneous polynomial continuation. *Numerische Mathematik Volume 66, Issue 1, pp 517-523*, 1993.

[82] Layne T. Watson, Stephen C. Billups, and Alexander P. Morgan. Algorithm 652: Hompack: A suite of codes for globally convergent homotopy algorithms. *ACM Transactions on Mathematical Software (TOMS)*, 13(3):281–310, 1987.

[83] Steven M. Wise, Andrew J. Sommese, and Layne T. Watson. Algorithm 801: Polsys_plp: A partitioned linear product homotopy code for solving polynomial systems of equations. *ACM Transactions on Mathematical Software (TOMS)*, 26(1):176–200, 2000.

[84] Alden H. Wright. Finding all solutions to a system of polynomial equations. *Mathematics of Computation*, 44(169):125–133, 1985.

[85] Wenyuan Wu and Greg Reid. Application of numerical algebraic geometry and numerical linear algebra to pde. *Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, 2006.

APPENDIX A
paramotopy INPUT FILES

INPUT A.1. Cube system

```
1 1 2 0
x^6 + y^6 + z^6 -1
z
0
x -1.5 0 1.5 0 200
y -1.5 0 1.5 0 200
```

INPUT A.2. Chambers of Real Solutions System

```
6 1 2 3
alpha*a1 - g*a1^3 + h1*(a2^2 + a3^2 + a4^2 + a5^2 + a6
   ^2)*a1 + h2*(a2*a3 + a4*a5) + h3*(a3*a4*a6 + a2*a5
   *a6)
alpha*a2 - g*a2^3 + h1*(a1^2 + a3^2 + a4^2 + a5^2 + a6
   ^2)*a2 + h2*(a1*a3 + a4*a6) + h3*(a3*a4*a5 + a1*a5
   *a6)
alpha*a3 - g*a3^3 + h1*(a1^2 + a2^2 + a4^2 + a5^2 + a6
   ^2)*a3 + h2*(a1*a2 + a5*a6) + h3*(a2*a4*a5 + a1*a4*
   a6)
alpha*a4 - g*a4^3 + h1*(a1^2 + a2^2 + a3^2 + a5^2 + a6
   ^2)*a4 + h2*(a1*a5 + a2*a6) + h3*(a2*a3*a5 + a1*a3*
   a6)
alpha*a5 - g*a5^3 + h1*(a1^2 + a2^2 + a3^2 + a4^2 + a6
   ^2)*a5 + h2*(a1*a4 + a3*a6) + h3*(a2*a3*a4 + a1*a2*
   a6)
alpha*a6 - g*a6^3 + h1*(a1^2 + a2^2 + a3^2 + a4^2 + a5
   ^2)*a6 + h2*(a2*a4 + a3*a5) + h3*(a1*a3*a4 + a1*a2*
   a5)
a1,a2,a3,a4,a5,a6
constant h1, h2, h3;
h1 = .53641376502734015;
h2 = .795362523521875;
h3 = .42332163652175211 78;
0
g -1 0 1 0 1000
alpha -1 0 1 0 1000
```

INPUT A.3. Chemical Reaction Network 1

```
7 1 2 21
Fa - Da*a - k4*a*c + k3*g + k2*f - k1*a*b
Fb - Db*b - k6*b + k5*c*d + k2*f - k1*a*b
Fc - Dc*c + k8*d - k7*c*e + k6*b - k5*c*d - k4*a*c + k3*g
Fd - Dd*d - k8*d + k7*c*e + k6*b - k5*c*d
Fe - De*e + k8*d - k7*c*e
Ff - Df*f - k2*f + k1*a*b
Fg - Dg*g + k4*a*c - k3*g
a,b,c,d,e,f,g
constant k3,k5,k6,k7,k8,k9,Fa,Fb,Fc,Fd,Fe,Ff,Fg,Da,Db,Dc,Dd
    ,De,Df,Dg,k1;
k3=.1180349;
k5=.8034;
k6=.8018;
k7=.16876;
k8=.7982;
k9=.58973;
Fa=.4264;
Fb=.5284;
Fc=.1687;
Fd=.167896;
Fe=.5673;
Ff=.69386;
Fg=.79827;
Da=.0692;
Db=.08762;
Dc=.2897;
Dd=.0828;
De=.26967;
Df=.4238;
Dg=.5872;
k1 = .42896979631;
0
k4    0 0 1 0 100
k2    0 0 1 0 100
```

INPUT A.4. Chemical Reaction Network 2

```
7 1 2 21
Fa - Da*a - k4*a*c + k3*g + k2*f - k1*a*b
Fb - Db*b - k6*b + k5*c*d + k2*f - k1*a*b
Fc - Dc*c + k8*d - k7*c*e + k6*b - k5*c*d - k4*a*c + k3*g
Fd - Dd*d - k8*d + k7*c*e + k6*b - k5*c*d
Fe - De*e + k8*d - k7*c*e
Ff - Df*f - k2*f + k1*a*b
Fg - Dg*g + k4*a*c - k3*g
a,b,c,d,e,f,g
constant k6,k7,k8,k9,Fa,Fb,Fc,Fd,Fe,Ff,Fg,Da,Db,Dc,Dd,De,Df
    ,Dg, k1, k2, k4;
k6=.8018;
k7=.16876;
k8=.7982;
k9=.58973;
Fa=.4264;
Fb=.5284;
Fc=.1687;
Fd=.167896;
Fe=.5673;
Ff=.69386;
Fg=.79827;
Da=.0692;
Db=.08762;
Dc=.2897;
Dd=.0828;
De=.26967;
Df=.4238;
Dg=.5872;
k1 = .42896979631;
k2 = .889426497621;
k4 = .8168659797357;
0
k3    0 0 1 0 100
k5    0 0 1 0 100
```

INPUT A.5. Chemical Reaction Network 3

```
7 1 2 21
Fa - Da*a - k4*a*c + k3*g + k2*f - k1*a*b
Fb - Db*b - k6*b + k5*c*d + k2*f - k1*a*b
Fc - Dc*c + k8*d - k7*c*e + k6*b - k5*c*d - k4*a*c + k3*g
Fd - Dd*d - k8*d + k7*c*e + k6*b - k5*c*d
Fe - De*e + k8*d - k7*c*e
Ff - Df*f - k2*f + k1*a*b
Fg - Dg*g + k4*a*c - k3*g
a,b,c,d,e,f,g
constant k6,k7,k8,k9,Fa,Fb,Fc,Fd,Fe,Ff,Fg,Da,Db,Dc,Dd,De,Df
    ,Dg, k1, k2, k4;
k6=.8018;
k7=.16876;
k8=.7982;
k9=.58973;
Fa=.4264;
Fb=.5284;
Fc=.1687;
Fd=.167896;
Fe=.5673;
Ff=.69386;
Fg=.79827;
Da=.0692;
Db=.08762;
Dc=.2897;
Dd=.0828;
De=.26967;
Df=.4238;
Dg=.5872;
k1 = .42896979631;
k2 = .889426497621;
k4 = .8168659797357;
0
k3    .005 0 .015 0 100
k5    0 0 .6 0 100
```

INPUT A.6. Chemical Reaction Network 4

```
7 1 2 21
Fa - Da*a - k4*a*c + k3*g + k2*f - k1*a*b
Fb - Db*b - k6*b + k5*c*d + k2*f - k1*a*b
Fc - Dc*c + k8*d - k7*c*e + k6*b - k5*c*d - k4*a*c + k3*g
Fd - Dd*d - k8*d + k7*c*e + k6*b - k5*c*d
Fe - De*e + k8*d - k7*c*e
Ff - Df*f - k2*f + k1*a*b
Fg - Dg*g + k4*a*c - k3*g
a,b,c,d,e,f,g
constant k6,k7,k8,k9,Fa,Fb,Fc,Fd,Fe,Ff,Fg,Da,Db,Dc,Dd,De,Df
    ,Dg, k1, k2, k4;
k6=.8018;
k7=.16876;
k8=.7982;
k9=.58973;
Fa=.4264;
Fb=.5284;
Fc=.1687;
Fd=.167896;
Fe=.5673;
Ff=.69386;
Fg=.79827;
Da=.0692;
Db=.08762;
Dc=.2897;
Dd=.0828;
De=.26967;
Df=.4238;
Dg=.5872;
k1 = .42896979631;
k2 = .889426497621;
k4 = .8168659797357;
0
k3    .003 0 .03 0 100
k5    .003 0 .03 0 100
```

APPENDIX B
galeDuality MANUAL

B.2.1. INTRODUCTION TO GALEDUALITY

The program `galeDuality` by Dan Bates, Jonathan Hauenstein, Matt Niemerg, and Frank Sottile implements algorithms based on [12], [16]. This manual provides information on how to use the software while the aforementioned articles provide more of the mathematical theory behind the algorithms. We provide compilation instructions, sample input files, and information regarding all of the menu choices. The software symbolically computes the gale functions of the associated fewnomial system or vice versa. If prompted, the solutions of the system can be displayed as well.

B.2.2. COMPILING GALEDUALITY

galeDuality is written in C++ and requires the use of of three open-source software packages in order to run correctly. These packages are GAP [33], Bertini [13], and alphaCertified [43]. See the corresponding websites for each package maintained by their respective authors as to how to properly install the software. The galeDuality software has the capacity to run the Khovanskii-Rolle Continuation scripts (included in this tarbell) for systems in which the number of monomials exceeds the number of variables by only 2. This functionality requires the use of the proprietary computer algebra system software known as Maple. The user does not have to use the Khovanskii-Rolle Continuation maple scripts, unless desired. The complete symbolic transformation can be performed without running these scripts.

After the installation of each of the software packages, they all must be placed in a directory of the $PATH environment variable, and the gap executable must be named "gap". A gnu MAKEFILE is included and should require no additional modifications, assuming that the precautions in this paragraph are adhered to.

After 'untarring' the tarbell, the user can make the program by moving into the source directory and then type 'make galeDuality' in the command line.

B.2.3. Using `galeDuality`

To use galeDuality, the user must create an appropriate input file that corresponds to either a fewnomial system or a gale system. There are no additional configuration settings the user needs to provide before running the main program. `galeDuality` will ask the user to choose several options as to how to display the dual of the input system, create the dual system, or display the solutions of the dual system. In addition, the user will be prompted on whether the user wants to use `alphaCertified`[43] or not on the computed solutions of the dual of the input system.

B.2.3.1. Input for Fewnomial Systems. The first line of the file indicates the excess of the number of monomials over the number of variables, i.e. if the number of monomials is 9 and the number of variables is 4, then number placed on this line is 5. The second line indicates the dimensionality of the components that correspond to the solutions of the fewnomial system. Currently, the software only works for the zero-dimensional case. The third line indicates the number of variables in the fewnomial system.

The next line is a list of the monomials, delimited by spaces. Exponentiation is indicated by the "$\wedge$". In addition, if an exponent is negative, it must have encapsulated on either side by either a pair of "()" or a pair of "{}". Multiplication of variables is indicated by the "*" character. Concatenation does not suffice. For example, if the monomial is $a^2b^{-1}c^4$, the corresponding input for this monomial would be either "$a \wedge 2 * b \wedge \{-1\} * c \wedge 4$" or "$a \wedge 2 * b \wedge (-1) * c \wedge 4$". The constant monomial is written as 1.

The next lines are the leading coefficients of each of the monomials in the fewnomial system, written as rationals. For example, if one of the functions in the fewnomial system is

$$f(a,b,c,d,e) = a^2b^3c^{-4}e^{-2} + \frac{1}{3}a^{-1}b^2d^{-3} + \frac{2}{5}d^2e^{-5}$$

then the leading coefficients of $1, \frac{1}{3}$, and $\frac{2}{5}$ would appear in the appropriate column that corresponds to the monomials $a^2b^3c^{-4}e^{-2}, a^{-1}b^2d^{-3}$, and d^2e^{-5}, respectively.

Suppose we have the following fewnomial system.

$$f_1(a,b,c,d,e) = a^{-1}b^2c^2d - \frac{1}{2}b^2c - 2b^{-4}c^{-7}d^{-5}e^{-1} + 1$$
$$f_2(a,b,c,d,e) = ac - \frac{1}{8}b^2c + \frac{1}{2}b^2c - 2b^{-4}c^{-7}d^{-5}e^{-1} - \frac{1}{2}$$
$$f_3(a,b,c,d,e) = bc^4d^4 + \frac{1}{16}b^2c + \frac{3}{4}b^2c - 2b^{-4}c^{-7}d^{-5}e^{-1} - \frac{3}{2}$$
$$f_4(a,b,c,d,e) = d + \frac{3}{8}b^2c + b^2c - 2b^{-4}c^{-7}d^{-5}e^{-1} - 4$$
$$f_5(a,b,c,d,e) = e + \frac{1}{2}b^2c - 2b^2c - 2b^{-4}c^{-7}d^{-5}e^{-1} - 3$$

For the appropriate input file of this system, see the file "heptagon_102.dat" located in the examples folder.

B.2.3.2. Input for Gale Systems. The first line of this file indicates the number of variables (call this v) and the second line indicates the number of linears of a gale system (call this l). The next v lines are some basis of the null space of the monomial support matrix of the appropriate dual of this gale system. The monomial support need not necessarily be known. Essentially, there will be v lines, delimited by spaces, with l coordinates. The next line is a space delimited list of the variables followed by a 1 (to indicate the constant monomial). The next lines are the leading coefficients of each of the terms in the appropriate linears that form the positive chamber of the hyperplane arrangement of the gale system written as rationals.

Suppose we have the follow gale system.

$$m_1(s_0,s_1) = \left(\frac{-10}{11}s_0 + \frac{30}{11}s_1 - \frac{10}{11}\right)\left(s_0 - s_1 - \frac{1}{2}\right)^3 s_0 - (s_0 + s_1 - 1)s_1^3$$
$$m_1(s_0,s_1) = (s_0 + s_1 - 1)^3 s_0^2 - \left(\frac{-10}{11}s_0 + \frac{30}{11}s_1 - \frac{10}{11}\right)^2\left(s_0 - s_1 - \frac{1}{2}\right)s_1^2$$

For the appropriate input file of this system, see the file "pentagon_20" located in the examples folder.

B.2.3.3. Running galeDuality. galeDuality is a menu-driven program requiring either options inputted by the user for various choices or an appropriate file that can be redirected as input to the compiled C++ program that mimic input that would normally be provided by the user. The first choice is to indicate the type of input file, either a fewnomial or a gale system. The second choice is whether or not to display the solutions of the dual or the symbolic transformation of the dual itself.

The third choice determines whether or not to use Khovanskii-Rolle to solve the gale system or to use standard homotopy continuation methods via Bertini to solve the appropriate dual and will only be prompted if the previous choice of the user was a display of the solutions to the dual. The next option is whether or not to use the parametrization of the monomials with or without the heuristic for choosing a 'good' parametrization. The last option the user provides indicates whether or not to use the basis vectors generated by GAP or to use the heuristic for the 'best' sign patterns.

If the solutions are to be displayed, galeDuality will ask the user the max number of times to refine the solutions using the Newton Iteration algorithm provided in alphaCertified and will perform up to this number of Newton Iterations until the system is certified using floating point soft certification initially. If the solutions are all certified using floating point operations, galeDuality will ask the user whether or not to perform a rational hard certification of the soft certified solutions.

B.2.4. Output of galeDuality

galeDuality does not write any files to the disk, but will rather display the appropriate solutions requested or display the appropriate dual of the input system. Depending on various choices provided by the user, galeDuality will display on the console, the output from the Maple script and alphaCertified. If the solutions are displayed, alphaCertified will produce its appropriate output files and one may find the user's choice of solutions to be displayed saved in the refinedPoints file (see the alphaCertified documentation for additional details of files generated by alphaCertified). The additional coordinates of the solutions

found in this file can be safely ignored, as these coordinates correspond to changing the laurent system to an appropriate polynomial system with the same solutions, in the case the user chooses to display the fewnomial solutions. In the situation where the user chooses to display the solutions to the gale system, the first n coordinates may be safely ignored as these correspond to the solutions of the linears and not to the variables of the gale system.